Haute Ecole Libre de Bruxelles Ilya Prigogine
Département technique HELB INRACI
Avenue Victor Rousseau, 75
1190 Bruxelles

Développement d'un système embarqué de détection rapide de précipitations et d'échantillonnage de l'eau de pluie

Directeur de stage et de TFE : M. Jean Morren
Maître de Stage en entreprise : M. Maxime Duvivier

Année académique : 2009-2010 **Thibault De Groef**

Mémoire présenté en vue de l'obtention du titre de bachelier en Electronique

Remerciements

Je tenais à remercier particulièrement M. Jean Morren, pour la proposition de stage au C.R.A., son aide et encadrement technique, ainsi que son dévouement quand à la réalisation du projet.

Je remercie ensuite M. Maxime Duvivier et M. Jean-Marc Moreau, du Centre de Recherches Agronomiques de Gembloux, pour l'accueil, le sujet de TFE, leur confiance envers nous, et leur disponibilité.

Je voudrais également remercier M. Pascal Tarte, qui a réalisé la majeure partie mécanique de notre projet. Sa disponibilité a toujours été au rendez-vous, malgré plusieurs problèmes de planning (M. Tarte travaillant dans une autre équipe que la nôtre).

Tables des matières

Introduction 6
Cahier des charges 9
Choix technologiques 10
  Microprocesseur 10
  Détection début de pluie 11
  Moteurs 14
  Sauvegarde de données 16
  Composants électroniques 17
    Régulateurs 17
    Horloge RTC 19
    L'extendeur de ports 19
    Les mémoires 20
    Le Timer 555 / 556 21
    Sensirion : Capteur d'humidité et de température 21
  Le panel utilisateur 22
1. Partie Hardware : Schéma bloc 23
  ADµC 832 23
  Alimentations 24
  Capteur de pluie 25
  Les sources d'interruptions extérieures 28
    Le pluviomètre 30
    L'horloge RTC 31
    Ouverture du capteur 31
    Le panel utilisateur 31
  Encodeur de position absolue 32
  Barrière infrarouge 33
  Les Servomoteurs 35
  Service USB : VDIP2 35
    VNC1L : Schéma bloc 36
    Les firmwares 38
    L'interfaçage 39
    Le format FAT 16 41
      Structure d'une FAT 16 : 42
Circuit imprimé - Principal 46
  Soudage du PCB 47
  Relevé des consommations 49
Circuit imprimé : Barrière infrarouge 50
Problèmes rencontrés 51
Récapitulatif des éléments rendant notre projet défensif point de vue Hardware 53
2. Programmation 54
  I. Analyse du programme 54
    Programmation embarquée 54
    Interface de paramétrage 62
  II. Description modulaire 63

Getposition 63
SendEdata 63
Savelog 63
Movemerrygoround 63
OpeningCouvercle 64
SerialInterrupt 64
TICinterrupt 66
Convad10 68
Calcultemp 68
Initialisation 68
Scrutint 69
Inter0 71
RTCI 72
USBI 76
Pluvioi 78
Serviceintopi 80
VDIPION 81
OPI 81
Servicemainint 83
Main 84
III. Programmation du module spécifique : VDIP2 85
IV. Eléments rendant notre projet défensif point de vue software et problèmes rencontrés 93
3. Partie mécanique 94
La structure du mécanisme 94
Système de changement de filtres 95
Les tubes et le système de fixation des filtres 98
Barrière infrarouge 102
Le panel utilisateur 105
Couvercle 107
Problèmes rencontrés 108
Servomoteur 108
4. Tests réalisés 109
5. Mode d'emploi 110
Mise en service : 110
Connexion entre le PC et la carte 110
6. Bref rappel des protocoles utilisés 113
Le Bus I2C 113
Le bus USB 115
Introduction 115
Le protocole 116
Types de transferts 117
L'UART 118
7. Améliorations possibles et prévues 120
8. Liste des composants et devis 122
9. Conclusion 124
10. Bibliographie et annexes 125

Introduction

Contexte

Certains champignons sont capables de causer des pertes de rendement très importantes sur des cultures telles que le froment d'hiver. Ces phytopathogènes se disséminent souvent à l'aide de spores (*Figure 1*) qui peuvent voyager dans l'atmosphère sous forme *d'inoculum* aérien. Les effets de *l'inoculum* aérien sont beaucoup dévastateurs s'ils sont véhiculés par la pluie que le vent.

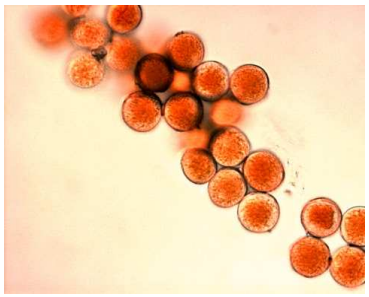


Figure 1 : spores de Puccinia Striformis, agent de la rouille jaune du blé

Objectifs du projet

Le but est de quantifier ces spores afin de connaître l'impact de celles-ci sur les cultures et de déterminer au fil des saisons quand mettre les bons fongicides. Cet apport de spores est bien plus important, et dévastateur que l'apport de spores par le vent par exemple.

L'évolution des conditions climatiques et les récentes observations sur les pathogènes et leurs contrôles suscitent actuellement des questions pertinentes quant à l'origine et la dispersion spatio-temporelle des *inoculums* aériens des maladies du blé aussi importantes que le *piétin-verse*, la *septoriose*, les rouilles jaune et brune ou les fusarioses.

Les objectifs du travail sont de mettre en place sur l'ensemble de la zone de grande culture Wallonne un réseau de capteurs de spores fongiques, de caractériser qualitativement et quantitativement les spores ainsi collectées au moyen d'outils moléculaires très sensibles et performants et de mettre en relation les informations recueillies selon cette méthodologie avec le développement effectif des maladies en champs. Ceci, dans le but d'évaluer l'incidence des *inoculums* aériens sur les maladies et de développer des nouvelles stratégies préventives de lutte basées sur la caractérisation des *inoculums* aériens.

Pour y arriver, la stratégie se divise en trois axes complémentaires :

- Le développement de méthodologies pour échantillonner l'inoculum aérien et ce, par différents moyens :

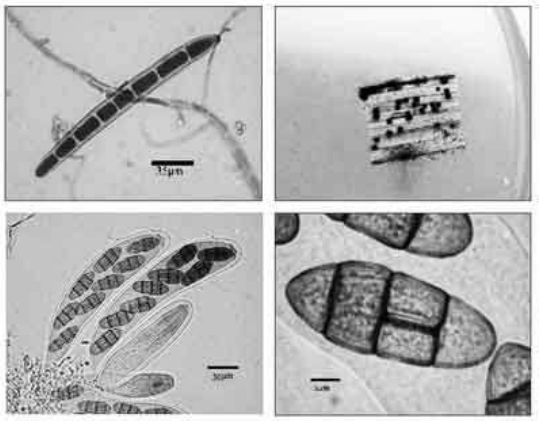
    - La mise en place et la gestion d'un réseau de 10 capteurs de spores répartis à travers toute la zone wallonne de culture de blé, qui puissent fonctionner sans discontinuer, de manière à disposer d'une banque spatio-temporelle originale de spores collectées dans l'air.

    - La mise au point de capteurs capables de filtrer la pluie afin de détecter les spores lessivées par les précipitations.

    - Le développement d'un protocole pour détecter et quantifier l'inoculum aérien effectivement déposé sur les plantes de la culture.

- Le suivi du développement des maladies en champs en relation avec la localisation géographique, les conditions climatiques, la variété et la stratégie de traitements fongicides.

- Le développement d'une méthodologie permettant l'extraction du matériel génétique à partir des échantillons fournis par les capteurs de spores, par les capteurs de "pluie" et des échantillons récoltés dans la culture, ainsi que la mise au point de méthodes de détection et de quantification moléculaires spécifiques des principaux pathogènes du blé.



Une étude similaire est en cours aux USA, mais celle-ci est moins précise que celle entreprise au CRA-W. En effet, l'étude menée ici portera sur les spores lessivées, avec une quantification journalière, alors qu'aux USA, cette quantification est aléatoire et se compte en semaines.

Résumé global et succinct du projet

Les précipitations peuvent laver l'atmosphère et amener les spores de pathogènes sur les cultures dans des bonnes conditions de germination. Une étude de Li *et al.* (2008) sur la rouille du soja démontre que l'arrivée des urédospores par la pluie sur la culture pourrait être beaucoup plus importante que le dépôt par les vents de l'inoculum aérien. Aux Etats-Unis, l'utilisation d'un réseau d'échantillonneurs de pluie couplé à la PCR (*Polymerase Chain Reaction*) quantitative se révèle être un outil efficace pour prédire les risques d'épidémies de rouille noire et de rouille du soja (Barnes *et al.*, 2005, 2009).

Mots clés : spores, fongicides, rouille, soja, précipitations

Korte Inleiding

Regen kan lucht « wassen » en kan pathogene sporen op kulturen die in goede ontkiening condities zijn. Een studie van *Li et al (2008)* over soja roest toont dat het aankomst van uredospores door regen over velden, zou veel hoger liggen dan wat wind van de innoculum in de lucht zou spreiden. In de V.S. ; het gebruik van een netwerk van regensamplers, gelinkd aan een kwantitatief PCR (*Polymerase Chain Reaction*), heeft getoond een schitterend werktuig te zijn om het risico van epidemiën tegen zwarte roest en sojaroest voort te spellen. *(Barnes et al., 2005, 2009)*

Trefwoorden: sporen, fungiciden, roest, sojabonen, regen

Abstract

Rainfall can clean the atmosphere and bring pathogen spores on cultures in good germination condition. A study of *Li et al (2008)* about rust of soja, shows that uredospores transported by rain over cultures could be much more important than the deposit by the wind of the inoculum. In the U.S. ; the use of a rainsamplers network, linked to a quantitative PCR (Polymerase Chain Reaction), shows to be an efficient tool to predict the risk of an epidemia of black rust and the soja rust. *(Barnes et al., 2005, 2009)*

Keywords : Spores, fungicides, rust, soybeans, rain

Cahier des charges

Le système doit être capable d'échantillonner ou de filtrer les précipitations. Cet outil permettra d'évaluer *l'inoculum* de pathogènes présent dans la pluie. Le système doit être autonome pendant au moins 7 jours. Il doit permettre d'obtenir des échantillons contenant la pluie d'une journée ou plus. Lorsque le système est employé pour filtrer la pluie, le filtre doit filtrer la pluie d'une journée ou plus.

Le système doit posséder un couvercle hermétique aux spores qui s'ouvre rapidement lorsque la pluie débute, et qui se referme après un laps de temps définis, lorsque la pluie est terminée.

En plus de ces caractéristiques de base le système permettra de mesurer la pluviométrie, l'humidité relative, le point de rosée, la température.

Les données météorologiques ainsi que les informations relatives au bon fonctionnement de l'appareil seront récupérables sur une clé *USB* sans nécessité de connecter un ordinateur, et exploitable sous Microsoft Excel.

Des paramètres tels que la sensibilité de l'ouverture du couvercle, le laps de temps pour un échantillon, le type de pluviomètre, l'heure, la date, … seront modifiables via une interface informatique simple d'usage.

Le système doit rester bon marché en vue de pouvoir créer un réseau de ce type d'appareils. Il sera donc aussi facilement reproductible.

Le système sera placé dans un champ. Il doit être robuste, étanche et résister aux intempéries. Le système doit être simple d'utilisation et la maintenance doit être aisée.

Le choix technologique est libre.

A la fin du stage, un prototype doit être fonctionnel et exploitable.

Choix technologiques

Nous n'avions pas ou peu de contraintes quand aux chois technologiques. Il nous a juste fallu tenir compte des températures de fonctionnement. Tous nos composants ont au minimum une gamme de température de -40° à +85°C, en fonctionnement.

Microprocesseur

Pendant nos années d'études à la HELB-INRACI, nous avons principalement étudié le 8051, et plus précisément, *l'ADµC 832 d'Analog Devices*. Le cahier des charges nous imposant implicitement beaucoup de broches d'entrées et de sorties, des entrées analogiques, éventuellement des sorties PWM pour la commande de servomoteurs, … *L'ADµC 832* s'est imposé.

Avantages	**Inconvénient**
Connaissance du processeur	Courant de sortie faible (besoin d'étage tampon)
Beaucoup de périphériques disponibles et intégrés	
Commencer très vite les tests car nous avons déjà une plaquette de développement HELB-*ADµC*.	
Consommation faible	
Beaucoup de programmes déjà abordés	

Tableau 1 : Points faible et forts de l'ADµC 832

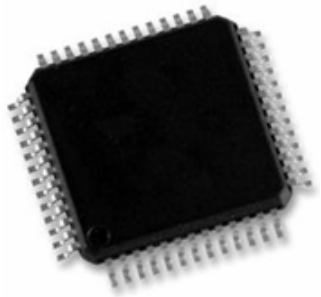


Figure 2 : L'ADµC 832

Le positionnement :
Plusieurs possibilités s'offrent à nous, parmi lesquelles :

- Encodeur de position absolue (gray)
    - Optique
    - Contacts mécanique
- Barrière infrarouge
- Contact mécanique « fin de course »

L'encodeur optique n'a pas été retenu pour des soucis de consommation, face à l'encodeur à contacts mécaniques qui présente le grand avantage de ne consommer que très peu (dépendant des résistances mises en jeu).

L'encodeur seul ne suffisant pas (en cause les dérives normales de la mécanique, dues à son jeu de fonctionnement), nous avons choisi d'intégrer une barrière infra rouge, commandée par notre microcontrôleur.
Le système fin de course n'a pas été retenu, car il n'est pas « ré-armable » !

Détection début de pluie

La contrainte est d'être réactif quand à la détection d'une pluie. Nous avons plusieurs hypothèses de travail :

- Utiliser le premier basculement d'un pluviomètre à auget basculeur
- Utiliser les propriétés conductrices de l'eau
- Utiliser la variation de permittivité de l'eau par rapport à l'air
- Utiliser les propriétés de réfraction de la lumière de l'eau.

Pluviomètre

Figure 3 : Pluviomètre PR12 implanté sur une station météo GmE

La cuillère (dit auget), contient 6ml. Dans le pire des cas, c'est-à-dire quand l'auget est vide et qu'il y a une pluie fine (1 mm/h), il faudrait 12 min (*avec le pluviomètre PR12 0,2 mm*) avant de détecter un seul basculement. Si l'on désire prendre de la marge et détecter deux ou trois basculements consécutifs, on aurait une très mauvaise sensibilité. C'est la raison pour laquelle nous avons abandonné cette solution

Propriétés conductrices de l'eau

Figure 4 : Capteur d'humectation à détection résistive

Méthode donnant des résultats satisfaisants dans la rapidité de détection. Un réseau d'entrelacements de deux électrodes est dessiné sur le dessus du capteur, lorsqu'une goutte tombe, elle fait contact et l'électronique interne actionne un relais.

Un chauffage est présent (150 à 200mA @ 12V) pour évaporer l'eau et donc réarmer le capteur, pour détecter une pluie suivante. C'est là son gros défaut, il n'est pas réarmable nativement, c'est-à-dire qu'il faut que la goutte faisant contact soit d'abord évacuée/évaporée.
Le prix de ce capteur (dit d'humectation), est d'environ 55€.

Variation de permittivité de l'eau par rapport à l'air

*La **permittivité :** plus précisément permittivité diélectrique, est une propriété physique qui décrit la réponse d'un milieu donné à un champ électrique appliqué.*

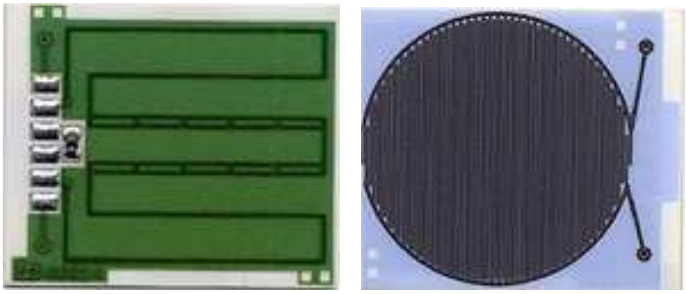


Figure 5 et 6 : Vue de dos et de face du capteur d'humectance à détection capacitive

Ce capteur est également un capteur d'humactance, possédant un réseau d'électrodes entrelacées. La capacité nominale (à l'air) est de 100pF (+/-10%). L'apport d'eau sur sa surface striée induit une variation de cette capacité nominale. Une variation de capacité est facilement transformable en variation de fréquence (via un astable), mesurable par un microprocesseur.

Après tests, le fonctionnement est très sain, une variation de plusieurs kHz, dû à l'apport d'une goutte, en un Δt très petit.

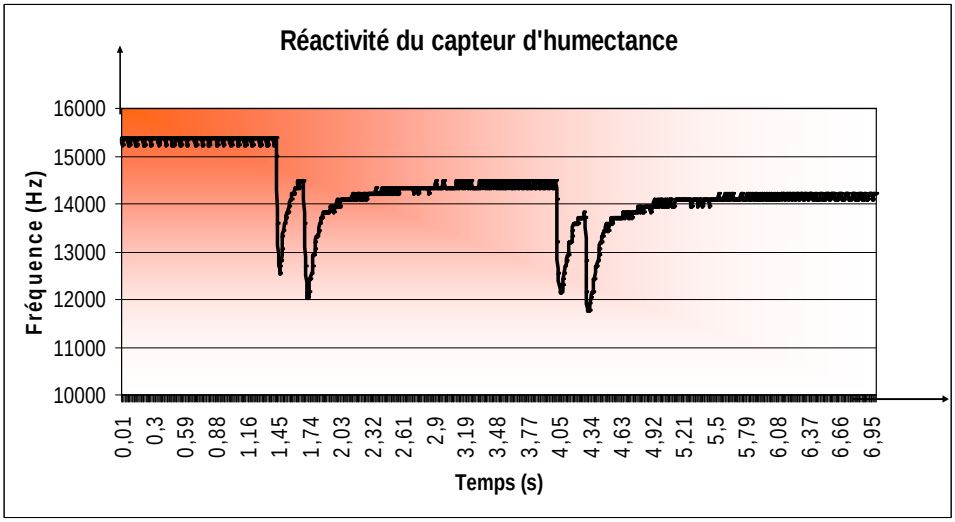


Figure 7 : Réactivité du capteur d'humectance lors d'apports de 4 gouttes d'eau

Pour ce test, quatre gouttes ont étés apposées, les quatre ont été détectées par le microprocesseur.

Il est donc nativement *réarmable*, tant que toute la surface n'est pas saturée en eau.

Néanmoins, la rosée a pour effet néfaste d'apporter de l'eau sur notre capteur, abaissant la fréquence de manière significative. Un chauffage est donc prévu (150mA @12V), pour réchauffer la surface du capteur si l'on s'approche du point de rosée (température pour laquelle il y a apparition d'eau sous forme liquide sur une surface), mais aussi pour évacuer l'eau amenée par la pluie au besoin (saturation, …)

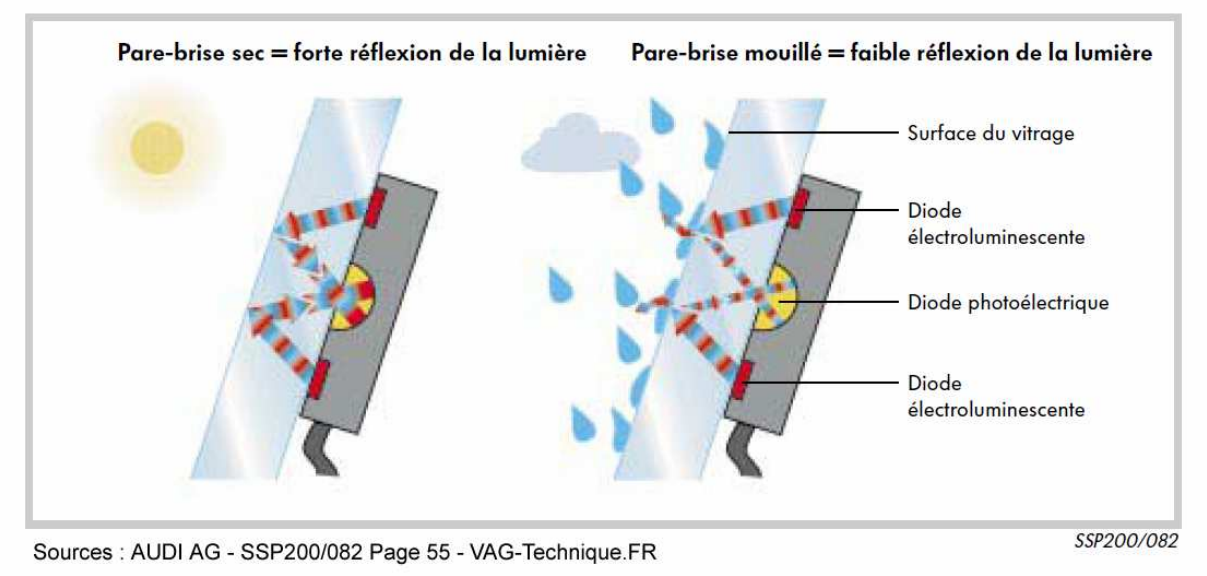
L'avantage d'une mesure de fréquence, c'est que celle-ci n'est pas influencée par le bruit.

Il faut incliner ce capteur à 30°, de façon à ce que l'eau puisse s'écouler par gravité.

Ce capteur ne coûte que 5.5€ HTVA, mais nécessite une petite électronique (un 555 monté en astable)

Utiliser les propriétés de réfraction de la lumière de l'eau.

Ce dernier procédé est celui utilisé dans l'industrie automobile. Notre capteur d'humectance donnant de résultats à la hauteur de nos attentes, cette technique n'a pas été testée.



Sources : AUDI AG - SSP200/082 Page 55 - VAG-Technique.FR

Figure 8 : Détection par réflexion du faisceau par une goutte

On distingue deux cas, le premier, sur la gauche, le pare-brise est sec, le capteur de pluie transmet un rayon de lumière au moyen des diodes électroluminescentes. Lorsque le pare-brise est sec, la totalité du rayon lumineux est réfléchi par la surface du vitrage. Si le pare-brise est mouillé, le rayon lumineux émis est réfléchi différemment. Une partie du rayon part vers l'extérieur. Au plus il y a de pluie (d'eau), au plus l'intensité lumineuse reçue est faible. De ce fait, un signal est envoyé au relais commandant les balais d'essuie-glace, balayant les gouttes d'eau.

Un tel système aurait imposé une commande d'essuie-glace, et aurait été peu pratique à mettre en œuvre.

Moteurs

Pour mettre en mouvement notre couvercle, et notre carrousel (*voir choix dans la partie mécanique*), il nous faudra deux moteurs de types différents ou similaires. Dans l'industrie, il existe une multitude de moteurs

- Pas à pas
- Courant DC
- Courant AC
- Servomoteurs
- …

Nous avons besoin de moteurs ayant beaucoup de couple, et ne tournant pas trop vite.

Nous avons donc réalisé une étude comparative des moteurs pas à pas par rapport aux Servomoteurs.

Moteur pas à pas

Avantages	**Inconvénients**
Précis	Consommation
Couple	Complexité de commande
	Nécessité de ressources matérielles (Circuits intégrés)
	Prix (45€)

Tableau 2 : Points forts et faibles d'un moteur pas à pas

Servomoteur

Avantages	**Inconvénients**
Facilité de mise en œuvre	180° de dynamique : corrigeable
Précis	Moins précis qu'un moteur pas à pas
Consommation raisonnable	
Couple de maintient natif (pas de courant) via la démultiplication	
Nous en avions directement pour les tests	
Prix (11€)	

Tableau 3 : Points forts et faibles d'un servomoteur

Le Servomoteur, pour le couvercle, ainsi que pour le carrousel a été choisi.

Dans un premier temps, nos tests ont été faits avec le *FUTABA* S3003, possédant un couple de 3,2kg.cm, soit 0,31Nm (@6V) pour un courant absorbé (mesuré) de ~ 450mA à son couple maximal.

Figure 9 : Servomoteur Futaba S3003

Le défaut d'un servomoteur réside dans le fait qu'on a une consommation inhérente de 10mA lorsque le servomoteur est sous tension, et non commandé (sans couple donc). Nous pouvons envisager de déconnecter l'alimentation avec un transistor de puissance mais on y perdrait en couple (VGS_{SAT} ⇔ 0V). On peut prévoir un régulateur dédié, 6V, actionnable par l'*ADμC*.

Sauvegarde de données

Le cahier des charges imposait de pouvoir récupérer les données enregistrées pendant la semaine d'acquisition sur une « clé *USB* ».

Nous avons trouvé chez le fabricant *FTDI* Chip, un module « kit de développement » *USB*, permettant le mode commandes (création de fichiers, gérant directement les *FAT* 12, *FAT16* et *FAT32*), mais aussi le mode *DATA* (*UART*, remplaçant le port série).

Nous avons dans un premier temps commandé le *VDIP1* chez *FTDI VINCULUM*. N'ayant qu'un seul port *USB,* l'utilisation en mode data était impossible (une des broches nécessaires n'étant pas disponible). Après recherches (car non disponible chez Farnell), il s'est avéré que le *VDIP2* existe, et est basé sur le même microcontrôleur, le *VNC1L*, et celui-ci permet directement le mode *DATA*.

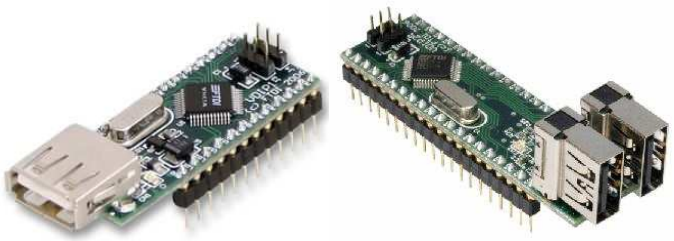


Figure 10 et 11 : VDIP1 et VDIP2 – module de sauvegarde de données par USB

La consommation en *powerdown* est de 2mA, en écriture de maximum 200mA, et en *idle* de 20mA.

Remarque : *pour tous les composants, nous avons à chaque fois opté au meilleur marché, quelle que soit la technologie (SMD, Trough Hole), et au moins consommant. Tous les composants sont commandés sur Farnell sauf exception clairement indiquée.*

Les plages de température de fonctionnement sont au minimum de -25° à +80°C.

Régulateurs

Comme expliqué précédemment, il nous faut 2 régulateurs (5 et 6V).

Les régulateurs à découpage ont retenu notre attention pour leur rendement. Nous avons, par choix, pris deux régulateurs ayant le moins de composants annexes (minimiser le risque de pannes, et faciliter un éventuel dépannage).

Nous désirons limiter le courant de sortie aux alentours de 350-450 mA pour le régulateur 5V.
Les deux régulateurs doivent supporter une tension d'entrée de minimum 25V (explication dans la partie schémas)

Les deux modèles retenus sont :

- *LT1111-5 pour le 5V*
- *LM2575 pour le 6V.*

Détails :

LT1111-5

Tension d'entrée	2 à 36V
Tension de sortie	5V fixe
Limiteur de courant	Via résistance
Courant maximal	1A
Maximum dissipation	500mW
Gamme de température	-40 à 105°C

Tableau 4 : principales caractéristiques de notre régulateur 5V

Il s'agit donc d'un régulateur à découpage pouvant être de type *step up* et *step down*. Nous l'utiliserons en *step down mode*.
La tension de sortie est comprise entre 4.75 et 5.25V

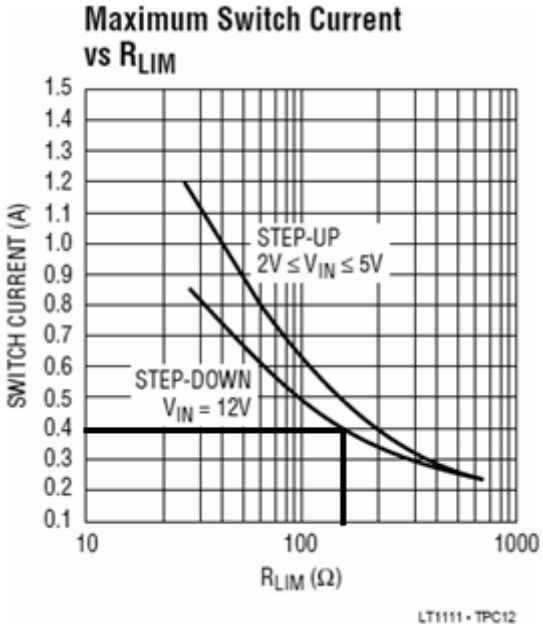


Figure 12 : Loi non linéaire liant la valeur de résistance en fonction du courant désiré du LT1111-5

Pour une protection du courant de sortie aux alentours de 400mA, nous devons mettre une résistance aux alentours de 170 Ω. Le boitier est un type SOIC8.

LM2576

Tension d'entrée	4.75 à 45V
Tension de sortie	1.23 à 37V +/- 4%
Tension d'activation	-0.3V à Vin
Limiteur de courant	3A fixe
Gamme de température	-40 à 125°C
Temps d'établissement (Off-On)	11.38 ms (10 à 90%, mesuré)

Tableau 5 : principales caractéristiques de notre régulateur 6V

C'est donc un régulateur *step down*, commandé par notre microcontrôleur, qui a une tension de référence (*feedback*) de 1.23V, nous devons donc choisir un rapport de résistance pour avoir notre tension en sortie (*6V*)

Boitier type *D2Pack*.

Pour les deux régulateurs, les valeurs des selfs, condensateurs et références des diodes sont celles prises aux « *typical applications* », des *datasheet*, correspondant à notre montage.

Horloge *RTC*

Nous avons besoin d'une horloge *RTC* pour faire, entre autres, des acquisitions synchrones, mais aussi pour horodater les évènements. Nous avons décidé, avec M. Morren, de fournir la pluie temps réel. De ce fait les centièmes de secondes sont indispensables. Seul *Phillips* propose cette fonctionnalité. Nous avons étudié au cours d'informatique la *PCF 8583*, mais celle-ci n'étant plus fabriquée en *SMD*, nous avons dû la choisir en *trought-hole.* La remplaçante *SMD* de la *PCF 8583* ne gérant plus les centièmes de secondes.

Figure 13 : PCF8583 Horloge RTC Phillips en I2C

Les avantages de cette horloge sont qu'elle se gère via le bus *I2C* de manière très aisée et qu'elle est autonome.

L'extendeur de ports

Bien qu'ayant 4 ports de 8 bits sur notre *ADµC* 832, un de ceux-ci est partagé avec les 8 convertisseurs analogiques-digitaux. Nous n'avons besoin que de deux (*voir dans la partie schémas*). Les 6 autres bits ne peuvent être utilisés comme entrées-sorties digitales en même temps.

Voyant le nombre de sources d'interruptions augmenter pendant le design, M. Morren nous a proposé un montage à l'aide du *PCF8574 IO Expander* (8bits).

Il se commande également sur le bus *I2C*, ce qui en fait un atout majeur lorsque l'on est critique en termes de ports d'entrées-sorties disponibles.

Ce circuit existant en *SMD* et en *Trough hole*, nous avons choisi le *SMD* pour son empattement moindre, à prix identique.

Figure 14 : PCF8574 IO Expander Phillips en I2C

Les mémoires

Afin de palier à d'éventuels problèmes de coupure de courant (changement de batterie sans pile de secours,…), nous plaçons les données dans des *RAM* non volatiles, afin de ne pas perdre ces dernières.

Notre *ADµC* possède une mémoire non volatile (dite *EDATA*) de 4kO (1024 pages de 4 octets), ce qui est relativement peu pour la sauvegarde de nos données. En effet, si l'on compte une moyenne réalisée sur une heure, comprenant diverses données :

- *Température minimale, moyenne et maximale*
- *Point de rosée minimale, moyenne et maximale*
- *Tension de la batterie en mV*
- *La date (hh :mm :ss DD/MM/YYYY)*
- *Débit minimal, moyen et maximal*
- *Cumul*

Nous arrivons facilement à plusieurs dizaines d'octets par heure d'acquisition, soit près d'1ko par jour (sans les Logs). L'autonomie d'une semaine ne peut être respectée.

Nous avons donc dû choisir une *RAM* non volatile externe. Nous avons commencé les investigations sur des *RAM* se connectant sur notre port d'adressage accessible (*A0-A15*).

Peu pratique à la réalisation (*PCB*, …), nous avons trouvé une alternative constituée de *RAM* non volatile interfacée sur le bus *I2C*. La facilité de commande est enfantine, tout ceci est géré via des librairies disponibles, tout en facilitant grandement le *PCB* (petit boitier *SOIC8*, avec deux pistes uniquement à router).

La référence choisie est la *RAMTRON FM24C512-G* en flash *RAM*, de *512kbits*, soit *64kBytes*. Nous en avons interfacé deux sur notre *PCB*, afin d'avoir plusieurs mois d'acquisitions possibles.

Cette mémoire possède deux pages de 32ko, il faut donc en tenir compte lors de l'adressage.

Le seul inconvénient de cette mémoire *RAM* est son prix, s'élevant à *15.33€ HTVA*.

Figure 15 : Ramtron FM24C512 – Mémoire non volatile I2C 64kB

Le Timer 555 / 556

Comme expliqué juste avant, nous avons placé notre capteur *IBR273* dans un astable à base d'un *555*. Lors du design de la carte, nous ne connaissions pas encore tout à fait la réactivité de ce capteur, ainsi que son comportement face à la rosée. L'idée première a donc été d'en associer deux à notre projet. Un pouvant être mouillé par la pluie, et son homologue, retourné, ne pouvant être influencé que par la rosée.

L'idée était donc que si un seul des deux capteurs avait sa fréquence modifiée, une pluie était discriminée. En revanche, si les deux fréquences variaient dans le même sens, c'était dû à la rosée. Nous avons donc décidé de mettre un *556* (double *timer 555*) pour compenser les dérives de fréquences dues à la température (que les dérives soient les mêmes, et donc dans le Δf elles soient nulles).

Nous avons dans un premier temps choisi le *LM556* et dessiné le *PCB* avec celui-ci. Après mesures, le *LM556* ne consommait pas loin de 3mA. Nous avons donc opté pour le *TLC556* pour sa faible consommation (400µA typiquement).

Les deux ont un fonctionnement identique et sont « *pin compatible* ». Le prix du TLC556 est de 1.19€ HTVA.

Figure 16 : TLC 556 dual Timer 555 faible consommation

Sensirion : Capteur d'humidité et de température

Pour l'algorithme de chauffage, nous avons intégré un capteur d'humidité et de température, le *Sensirion* SHT71, se branchant sur notre bus *I2C* (mais n'utilisant pas réellement ce protocole). Il nous permet d'obtenir facilement, et pour une consommation extrêmement faible (1mA maximum en mesures, et 1.5 µA en *powerdown*) la température et le taux d'humidité relative, au même point, ce qui nous permet de calculer le point de rosée.

Il peut être utilisé en 8 ou 12 bits (qualité de conversion, nous utiliserons 12bits). Les formules changent donc en fonction de la résolution. Il y a toutefois une grosse partie de calcul à faire, pour transformer la valeur brute en température et en taux d'humidité relative (HR). Pour des valeurs différentes de 25°C, il faut compenser le taux d'HR, en fonction de la température. Les formules sont présentes en annexes.

Nous utilisons les librairies de M. Morren, en *Assembleur*.

Ce capteur est calibré en usine.

Figure 17 :Sensirion SHT71 HR+t°

Le panel utilisateur

Dans le cahier des charges, il nous a été demandé de pouvoir récupérer les données par clé *USB*, et de paramétrer via une interface. Nous avons donc dessiné un panel fonctionnel disposé comme ceci :

- Un bouton poussoir demandant l'accès à un port *USB*:
  - Si une clé *USB* est présente ➔ Ecriture sur clé *USB*
  - Si un PC est présent ➔ passage en mode de communication PC ⇔ *ADµC*
- Une *LED* bicolore (rouge/vert) indiquant l'activité sur le port *USB* dédié à la clé *USB*
- Une *LED* Verte indiquant l'état de bonne santé de la batterie lors de l'ouverture du capteur uniquement
- Un bouton poussoir indiquant à l'*ADµC* que les filtres ont tous été changés

1. Partie Hardware : Schéma bloc

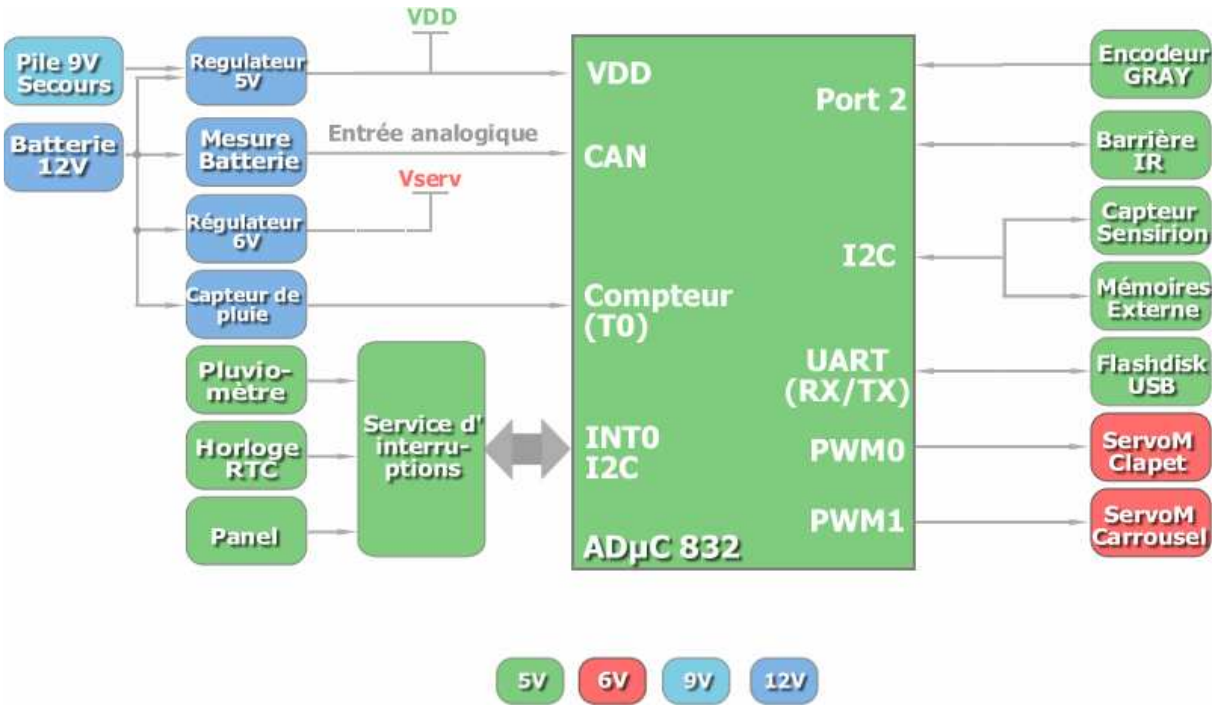


Figure 1.1 : Schéma bloc du projet

ADµC 832

L'ADµC 832 est donc notre microcontrôleur, cœur du système, basé sur l'architecture *INTEL* 8051.

Celui-ci possède beaucoup de périphériques, tels que :

- 8 convertisseurs analogiques digitaux 12bits
- 2 convertisseurs digitaux analogiques 12bits
- 2 sorties PWM 16bits
- 62 ko de Flash (remplaçant la ROM)
- 4 ko de RAM
- 4 ports 8 bits parallèle
- 3 timers/compteurs 16 bits
- Un UART, un bus I2C/SPI
- …

Un synoptique est disponible à la *figure 1.2*.

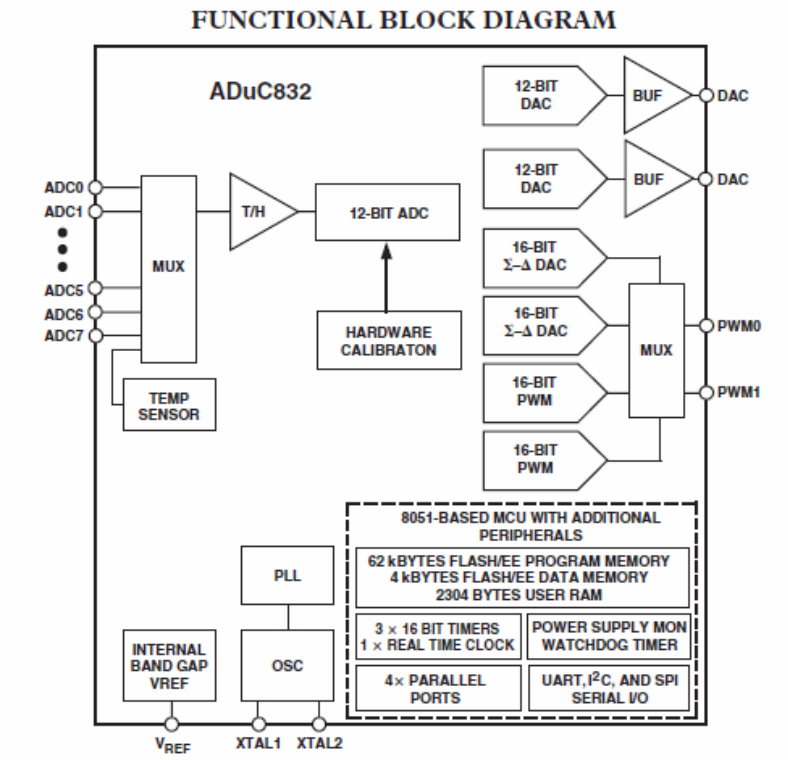


Figure 1.2 : Synoptique de l'ADµC 832

Les huit canaux de conversion analogique -> digital sont multiplexés, un seul est utilisable en même temps (de toute façon, un processeur ne peut faire qu'une action à la fois). Pour effectuer une conversion, il est conseillé de placer un filtre RC (10Ω, 100nF passe bas), afin de rejeter les composantes haute fréquence, mais aussi pour conserver la charge lors de petites fluctuations du *sample & hold* interne. *L'ADC* permet de s'auto-calibrer, en vérifiant sa tension de référence. Après conversion, deux registres contiennent la valeur de la conversion, allant de 0 à 4095.

Alimentations

Nous avons donc deux alimentations, 5 et 6V.

Nous devons mesurer la tension de la batterie, pour déterminer lorsque celle-ci est déchargée, afin de ne plus actionner le chauffage ainsi que les moteurs.

Par mesure de sécurité, et pour avoir une possibilité d'extension (panneau solaire), le système mesurant la tension est prévu pour une tension d'entrée de 25V, sachant qu'un panneau solaire à vide, peut fournir jusqu'à 21V .

Nous ne savons pas à l'heure du dessin du *PCB* si nous allons avoir une batterie propre au projet, ou si nous devront nous brancher sur une batterie présente sur le terrain. De ce fait, nous avons prévu un pont de *Graëtz* et un *bornier* en entrée, ainsi une erreur d'inversion de câbles est sans conséquences, mais nous « perdons » toutefois 2 x 0,6V (théorique, 0.95V mesuré, sans charges).

Nous avons également prévu une pile de secours (9V), alimentant uniquement le régulateur 5V (et donc le processeur et les circuits externes), n'étant pas mesuré par le processeur. Ce dernier mesure une tension soit inférieure à 9V, soit une tension indéterminée (entrée en l'air, si batterie débranchée). Il n'actionnera donc pas ni les moteurs ni le chauffage, préservant la durée de vie de la pile de secours. Les diodes de sélection de la source d'alimentation ont une chute de tension mesurée de 0.548V

Le régulateur 6V est actionné par notre processeur, le mettant en *power down*, et abaissant sa consommation à moins de 200µA (86µA mesuré)

Capteur de pluie

Il s'agit donc de *l'IBR273*, à variation capacitive, monté dans un 556, faisant varier la fréquence. Cette dernière est mesurée sur la broche *T0* de notre *ADµC* 832. Le 12V est relié au chauffage, et est commandé par l'*ADµC*.

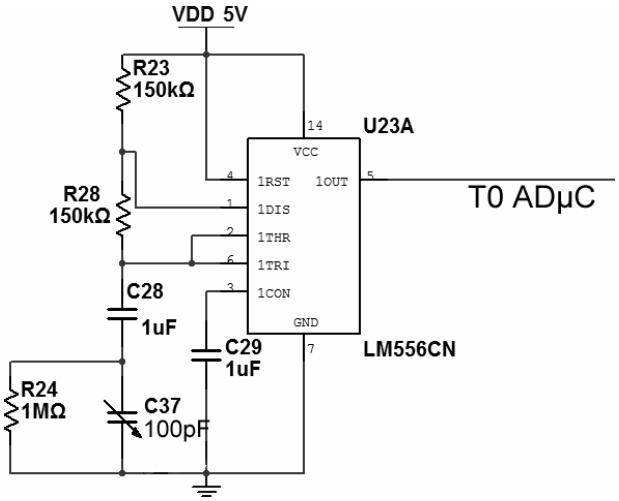


Figure 1.3 : Schéma du détecteur de pluie à variation capacitive

Avec ces valeurs de composants, la gamme de fréquence théorique est de :
Pour C = 100 pF à l'air, et jusque 7.8nF (permittivité relative maximal pour l'eau pure = 78)

$$f_{MAX} = \frac{1.44}{(R23 + 2R28).C37} = \frac{1.44}{(3.150.10^{3}).100.10^{-12}} = \frac{1.44}{3.150.100.10^{-9}}$$

$$= 32kHz$$

$$f_{MIN} = \frac{1.44}{(R23 + 2R28).C37} = \frac{1.44}{(3.150.10^{3}).7,8.10^{-9}} = \frac{1.44}{3.150.7,8.10^{-6}}$$

$$= 410,26Hz$$

Ces valeurs sont très théoriques. Dans la pratique, nous avons une dynamique allant de 15kHz à 7kHz, avec de l'eau non pure (robinet). Ces valeurs sont très dépendantes du capteur utilisé, en effet, ces dernières ont été mesurées avec un IBR273. Si on change d'IBR273, nous avons une dynamique un peu différente (14kHz – 7 kHz). C'est pour cette raison que nous allons détecter une variation franche (Δf) en un laps de temps très petit (Δt). Ainsi, on s'affranchit des dérives du capteur au fil du temps (modification de la dynamique, …), et on s'affranchit également du capteur utilisé.

Nous n'avons, à ce jour, pas réussi à descendre la fréquence de l'astable en dessous de 6kHz, avec de l'eau distillée empruntée au département chimie du C.R.A. – W.

La forme d'onde aux bornes du condensateur formant le capteur :

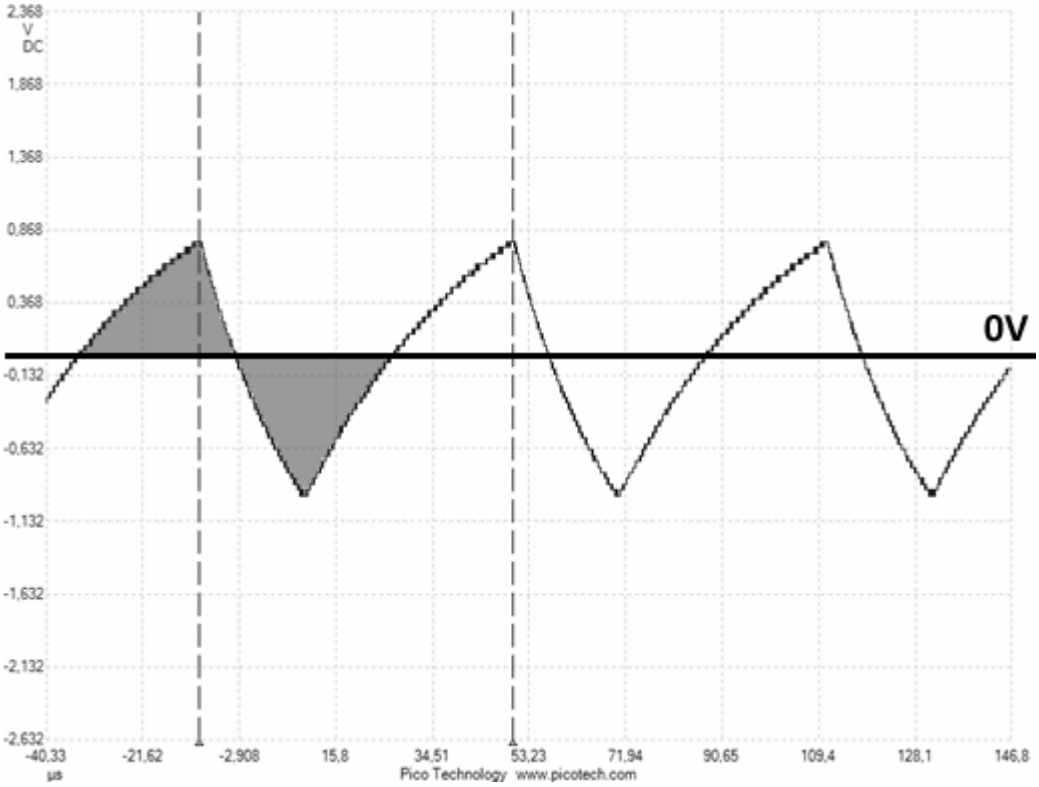


Figure 1.4 : Signal aux bornes du capteur de détection de pluie

La composante continue est donc bien nulle (pas d'électrolyse possible !)

En sortie de notre 556, nous avons un signal de type carré, dont le rapport cyclique est différent de 0.5, ce qui n'est aucunement grave pour notre application.

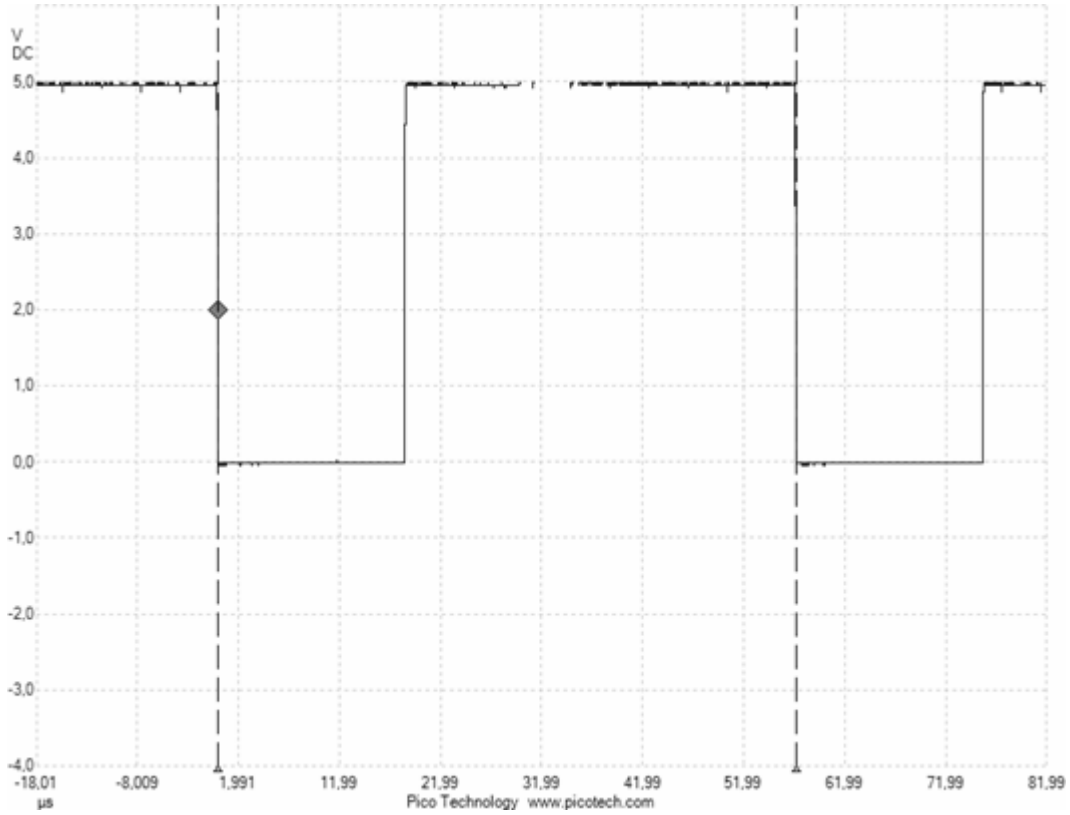


Figure 1.5 : Signal en sortie du 555

La fréquence de ces deux oscillogrammes est de 13696 Hz.

Les sources d'interruptions extérieures

Notre programme (qui sera expliqué dans la partie *Software*), est un programme qui n'est actif que lors d'interruptions.

Les sources d'interruptions sont gérées via un circuit sur le bus I2C

Remarque : Un rappel sur le bus I2C est disponible dans la section des rappels.

Chaque source est reliée à la broche int0 de *l'ADμC* 832, passant par l'intermédiaire d'un filtre anti-rebonds (RC), d'un inverseur et d'une bascule. Nous avons mis toutes les sources sur *int0*, et aucune sur *int1*, car au moment du design, on ne savait pas si on allait mettre en *powerdown l'ADμC*, et seul *int0* permet de le sortir de ce mode.

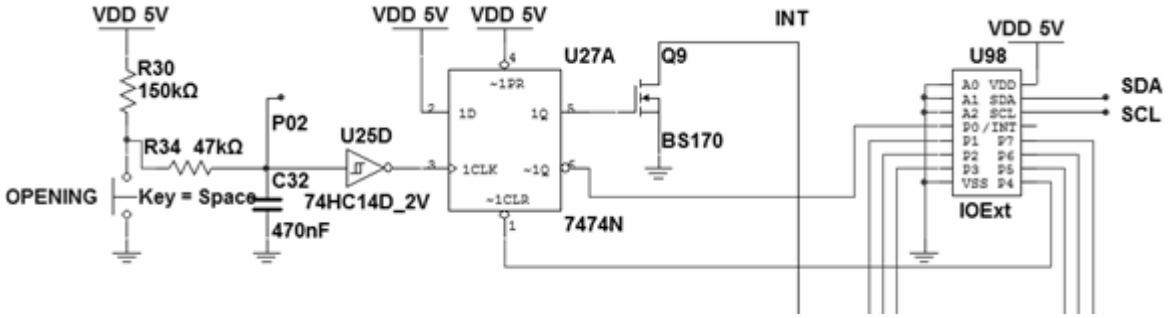


Figure 1.6 : schématique d'une des sources d'interruptions (x4)

La sortie de la bascule (mémorisant les diverses sources d'interruptions, si plusieurs arrivent simultanément) est connectée sur une broche du circuit *IO Expander* de *Phillips*, la *PCF8574*, qui est un port 8 bits, que l'on peut configurer indépendamment en entrée ou sortie, et qui se pilote via le bus *I2C*/*IIC*.

Lorsqu'une source d'interruption désire interrompre le microcontrôleur, un flanc montant est alors appliqué à l'entrée *CK* de la bascule, mémorisant l'état. La demande d'interruption est transmise via le transistor Q9 (pour cette source d'interruption) au μC.
Il sera alors désormais appliqué sur une des entrées de la *PCF8574* un niveau booléen bas. C'est ce niveau bas (et les autres hauts) qui détermineront la source qui demande l'interruption.
Si deux sources ont demandé en même temps l'interruption, ou qu'une deuxième a demandé pendant l'exécution de la première, le programme en sera informé grâce à la mémorisation des bascules. Lorsque toutes les sources d'interruptions ont été réalisées, le programme effectuera un reset des bascules via *l'IO Expander* (4 bits en sortie), réarmant les sources d'interruptions qui viennent d'être effectuées.

Chaque source possède son propre transistor permettant l'évolutivité du nombre d'interruptions (plus évolutif qu'une porte OU à quelques entrées figées)

La fréquence de coupure de nos filtres est de

$$f = \frac{1}{2\Pi RC} = \frac{1}{2\Pi.47.10^{3}.470.10^{-9}} = \frac{1}{2\Pi.47.470.10^{-6}}$$

$$= 7.20Hz$$

Tout rebond supérieur à 7.20Hz sera lissé par ce filtre passe bas.

L'efficacité du filtre a été mesurée

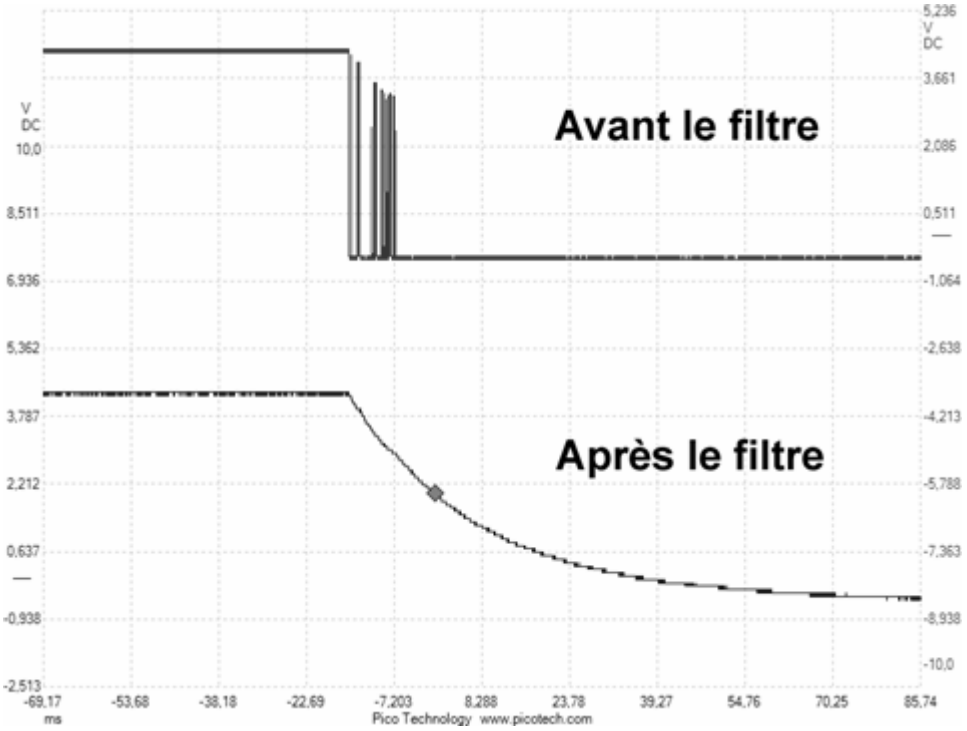


Figure 1.7 : Efficacité de notre filtre anti-rebond

On remarque donc que les nombreux rebonds sont gommés par le filtre, permettant de ne pas mettre en place un anti-rebond software, monopolisant du temps CPU.

Après le filtre se trouve une porte inverseur à entrée *trigger de Schmitt*, celle-ci permet d'éviter une oscillation du *clock* de la bascule.

Le pluviomètre

Nous avons intégré un pluviomètre à auget basculeur, permettant de quantifier la pluie.

Figure 1.8 : vue interne du type de pluviomètre choisi

Un pluviomètre est principalement composé d'un réceptacle et d'un auget basculeur. L'eau de pluie est collectée par le réceptacle et remplit l'auget goutte à goutte. Celui-ci bascule lorsqu'il contient 6ml d'eau, évacuant à chaque basculement 5 ml d'eau de pluie (il reste donc 1 ml dans l'auget). Plus deux basculements sont rapprochés, plus la précipitation est intense.

Sur le pivot de l'auget est disposé un aimant, lorsque celui-ci passe devant un *relais reed*, le contact se fait, on associe donc un pluviomètre à un interrupteur *NO* (ou *NF*, les deux sont disponibles sur le modèle que nous avons).

On interprète les résultats en mesurant le temps écoulé entre deux basculements consécutifs, et on peut déterminer l'intensité temps réel de la précipitation. Cette dernière s'exprime en volume d'eau par unité de surface (mm^3/mm^2) et est considérée comme exceptionnelle si le volume d'eau récolté pour une période définie, dépasse les normales relevées au même endroit.

	2 ans	5 ans	10 ans	25 ans	50 ans	100 ans
1 minute	1,7 mm	2,8 mm	3,7 mm	4,7 mm	5,4 mm	6,1 mm
10 minutes	8,2 mm	10,8 mm	13,2 mm	16,6 mm	18,5 mm	21.0 mm
60 minutes	14,9 mm	21.5 mm	26.0 mm	31,7 mm	35,8 mm	40.5 mm
1 jour	33,1 mm	43,2 mm	50,0 mm	56.2 mm	61.0 mm	67.0 mm

Tableau 1.1 : tableau représentant la fréquence d'apparition d'une pluie, pour un laps de temps donné

Le tableau ci-dessus représente la fréquence d'apparition d'une certaine pluie au centre de la Belgique. On le lit simplement : *Une pluie de 33.1 mm en moyenne sur la journée, revient tous les 2 ans, au centre de la Belgique*.

Le modèle choisi, « PR12 » est un pluviomètre 0.2mm, c'est-à-dire qu'il bascule pour $0.2\frac{mm^3}{mm^2} = 0.2mm$.

1mm correspond à 1 litre d'eau de pluie tombée sur 1m² (ou 10m³ d'eau de pluie tombée sur 1 ha)

L'horloge *RTC*

Nous avons intégré une horloge *RTC PCF 8584*, de *Phillips*, nous permettant d'avoir une base de temps fiable, mais également un « top » toutes les 10 secondes permettant d'effectuer des acquisitions synchrones de nos données.

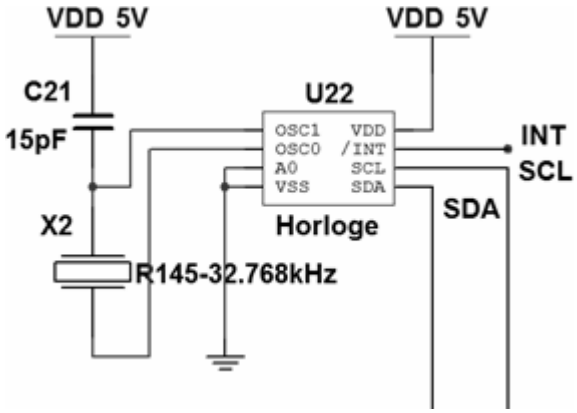


Figure 1.9 : Interfaçage de la PCF8583 Horloge RTC Phillips en I2C

C'est donc une source d'interruption programmée toute les 10 secondes, également présente sur le bus *I2C/IIC*.

La précision du Hertz est ajustable via le condensateur C21.

Ouverture du capteur

Lorsque le technicien viendra changer les filtres et récupérer les données, le système en sera averti par la fermeture d'un contact (*switch* fin de course). Le programme rentrera dans un service d'interruption, permettant, entre autre, l'initialisation du module *USB*.

Le panel utilisateur

Les deux dernières sources d'interruptions extérieures sont les deux boutons poussoirs du panel utilisateur. Le premier sert à demander soit le chargement des données sur la clé *USB*, soit à passer en mode de communication entre le PC et l'*ADμC* (selon celui qui est connecté), le deuxième sert à indiquer le changement des filtres.

Nous avons donc 4 fois le schéma présenté sur la *figure 1.6*, pour les 4 sources d'interruptions extérieures, en plus de l'interruption de la *RTC*.

Il y a aussi une source d'interruption interne, qui sera expliquée dans la partie soft.

Encodeur de position absolue

Pour déterminer quel est le tube en face du pluviomètre, nous avons intégré un encodeur de position absolue code GRAY.

Pour rappel, ce système permet de n'avoir qu'un bit de changeant d'une position à l'autre.
Par exemple, un système binaire de 4bits (16 positions), si une transition de la position 7 à 8 est effective, nous obtiendrons une modification du code 0111 allant à 1000.

Si jamais le positionnement n'est pas parfait, il se peut qu'on lise la position 0011 par exemple, qui n'a aucunement lien avec le changement opéré.

Le code Gray permet de combler cette erreur de lecture comme suit (pour 4bits/16 positions)

Position	Code	Position	Code	Position	Code	Position	Code
0	0000	4	0110	8	1100	12	1010
1	0001	5	0111	9	1101	13	1011
2	0011	6	0101	10	1111	14	1001
3	0010	7	0100	11	1110	15	1000

Tableau 1.2 : Couple Position et valeur lue codage GRAY

*On voit donc clairement que notre position 7 à 8 devient en code GRAY : **0**100 à **1**100, soit bel et bien un seul bit ne changeant. Et donc un risque de se tromper proche de zéro. La seule erreur possible étant de croire qu'on est resté sur la position sept, alors que l'on est physiquement sur la huitième.*

Comme déjà expliqué plus haut, nous avons choisi un encodeur de position à contacts mécaniques, pour des soucis de consommation. Le revers de la médaille étant une usure mécanique possible.

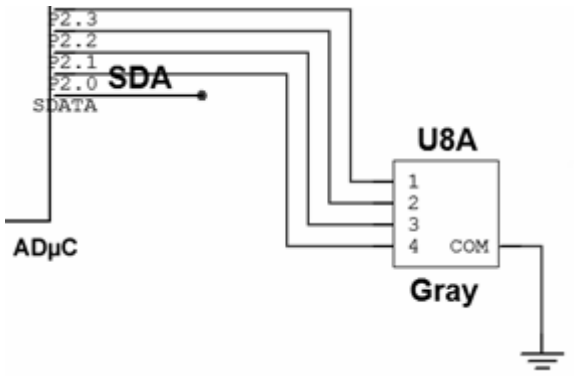


Figure 1.10 : Schéma de câblage de l'encodeur de position absolue – GRAY

L'encodeur gray 4bits est présenté sous la forme d'un potentiomètre, ayant 5 broches.

Figure 1.11 : Vue interne d'un encodeur GRAY 2bits

On peut le schématiser comme étant 4 interrupteurs :

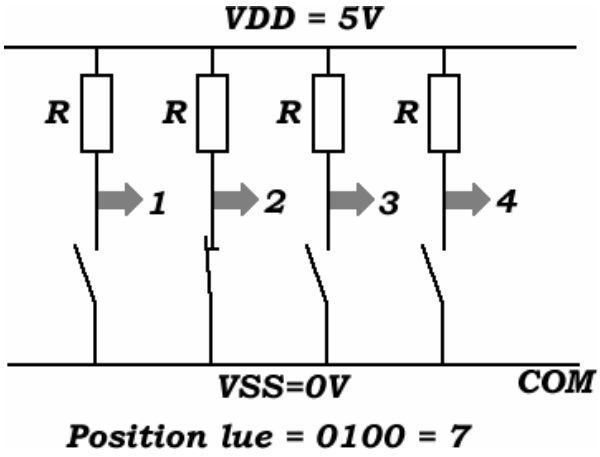


Figure 1.12 : schématisation d'un encodeur GRAY 4bits mécanique

Sur le schéma présenté *figure 1.10* on ne voit aucune résistance de *pull-up,* se sont en effet les résistances internes des ports de sorties de l'*ADμC* 832. Les points de connexion 1, 2, 3 et 4 sont les 4 broches (+ COM) disponibles au niveau de l'encodeur GRAY.

Barrière infrarouge

Comme déjà évoqué plus haut, la barrière infrarouge permet de combler les jeux ordinaires de la mécanique.

Nous avons donc choisi une *LED* Infrarouge, ainsi qu'un phototransistor répondant à ce même spectre. Notre *LED* est commandée via un transistor pour réduire la consommation lorsque l'on en n'a pas besoin (la majeure partie du temps).

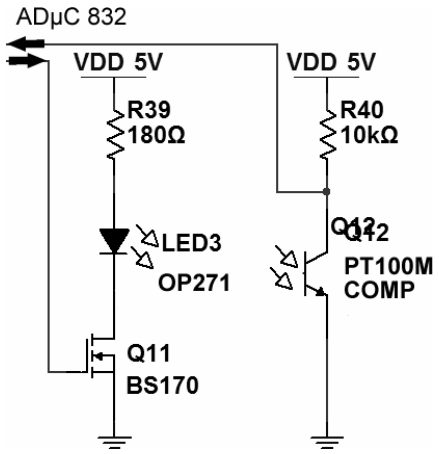


Figure 1.13 : Schéma de la barrière infrarouge

Le signal lu est soit bas, ce qui signifie qu'il n'a pas d'obstacle entre la *LED* et le phototransistor, soit haut, ce qui signifie que le faisceau lumineux est interrompu. Le courant traversant la *LED* est de 20mA, la tension à ses bornes est de 1.5V. La résistance a donc été calculée comme suit :

$$R_{39} = \frac{U_R}{I} = \frac{VDD - V_{LED} - V_{DS_{SAT}}}{I_{LED}} = \frac{5 - 1.5 - 0.04}{0.02} = 173\Omega$$

Nous avons placé une résistance de 180 Ω (série normalisée), pour un courant mesuré de 19.30mA

R40 est une simple résistance de *pull up* de 10kΩ

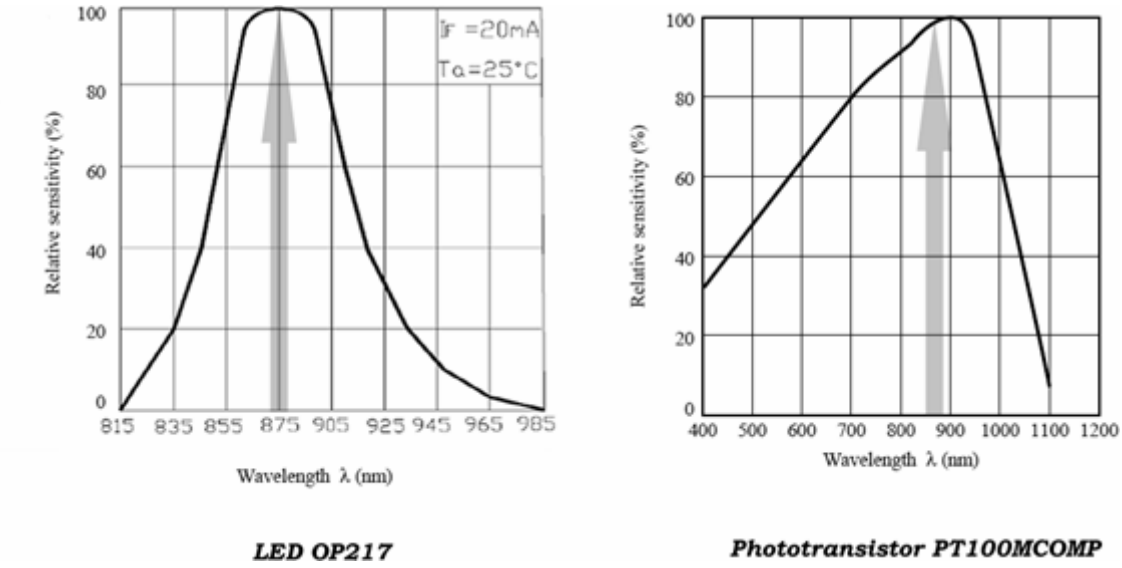


Figure 1.14 : diagramme de rayonnement / de réception des composants formant la barrière

En regardant les diagrammes de sensibilité spectrale pour le phototransistor, et celui de distribution spectrale pour la *LED*, on peut apercevoir que ceux-ci ne sont pas centrés à la même longueur d'onde. Mais, vu la grande fenêtre d'émission et de réception, et le petit écart entre les deux maximums (25nm), les deux bandes se chevauchent.

Les Servomoteurs

Les servomoteurs sont commandés par largeur d'impulsion. Une certaine largeur d'impulsion correspondant à un angle de rotation de ce dernier, si l'on respecte ces conditions :

- Fréquence de répétition des impulsions : autour de 50Hz (Optimum).
- Largeurs d'impulsions comprises entre 0 et 2.5ms.

Un Servomoteur est un moteur à courant DC asservi (position lue grâce à un potentiomètre), la rotation maximale est de 180°. Or, pour notre carrousel, nous sommes amenés à tourner plus d'un tour (démultiplication entre l'axe du carrousel et l'axe du moteur). Nous avons donc modifié ce dernier pour pouvoir effectuer une rotation continue dans un sens où l'autre, en remplaçant le potentiomètre par deux résistances formant la même valeur de butée que le potentiomètre. La valeur lue par l'électronique interne du Servomoteur étant toujours la même, nous pouvons tourner dans un sens si la largeur d'impulsion est supérieure à 1.25ms, et dans l'autre si cette-dernière est inférieure à 1.25ms.

L'inconvénient majeur est que l'on a perdu l'asservissement en position, et que nous n'avons plus la possibilité d'arrêter la rotation du carrousel en maintenant du couple (problème d'inertie).

Nous avons mis au point un algorithme logiciel pour la mise en rotation du carrousel, qui sera expliqué dans la section adéquate (page 63)

Par mesure de sécurité, nous plaçons un tampon en sortie de nos PWM présentes sur l'*ADμC*.

Pour des raisons de couple trop bas pour le couvercle, nous avons du changer de servomoteur, pour un *FUTABA* S9157, faisait 30,6kg.cm soit 3,01 Nm. La différence réside dans la commande se faisant à 300Hz au lieu de 50Hz. Tout le reste restant identique.

Service USB : VDIP2

Notre module *USB* constitue un des éléments principaux du projet. En effet, avec celui-ci, nous allons être capables de sauvegarder nos données d'acquisitions sur une clé *USB*, dans le format correspondant au formatage : *FAT 12*/*16* ou *32*. Ce module permet également l'émulation d'un port série, via *l'USB*, pour la programmation mais aussi pour le paramétrage des différents paramètres.

VNC1L : Schéma bloc

Le *VDIP2* est donc basé sur le microcontrôleur *VNC1L* :

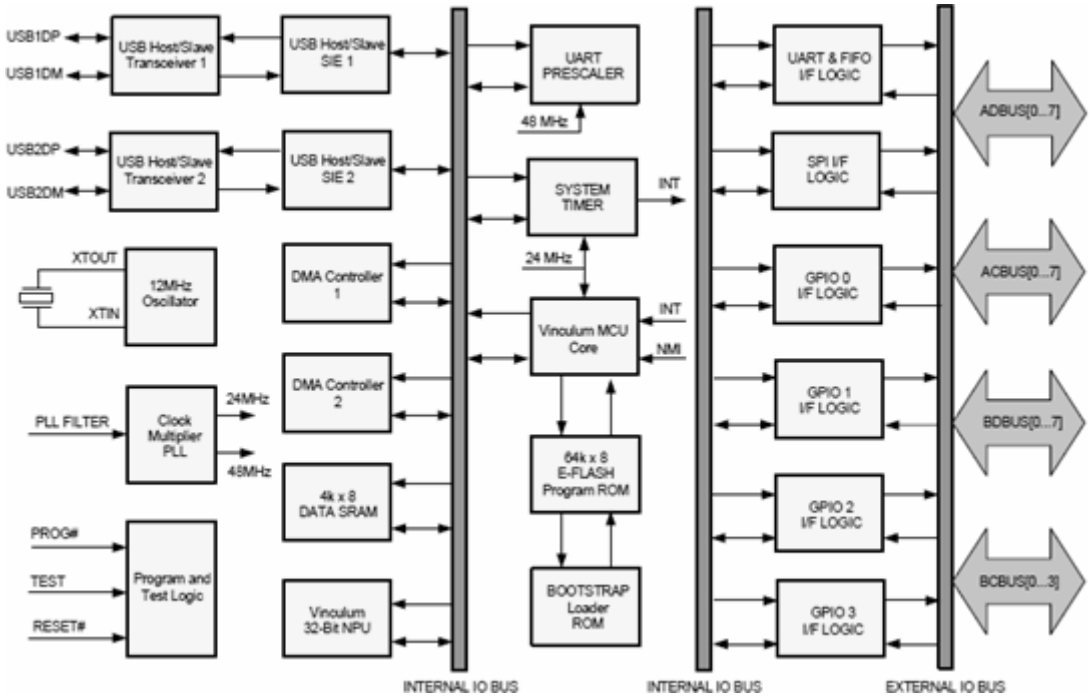


Figure 1.15 : schéma bloc interne du VDIP2

I. USB Tranceivers 1 and 2

Les cellules émetteurs-récepteurs *USB* (1 et 2) supportant la norme *USB* 1.1 et 2.0 (*rappel de la norme USB dans la section adéquate).* Chaque cellule pouvant gérer des vitesses hautes et basses de communication, et ayant un contrôle de *slew rate*. Ces cellules intègrent également les résistances de *pull-up* et *pull-down* requises pour les modes *slaves* ou *host*.

II. USB Sérial Interface Engine (SIE) :

Cellule qui gère la trame de l'*USB*, gère les conversions séries⇔parallèles, crée les CRC, …

III. Oscillator 12MHZ :

Référence de temps 12MHz

IV. Clock Multiplier PLL :

Multiplication du signal 12MHz fournis en amont. Peut générer du 24 et 48Mhz, requis par *l'USB SIE*, le *MCU* et *l'UART Prescaler*.

V. Program and test logic :

Cellule permettant de fournir un moyen de programmation de la mémoire *E-Flash*. Lorsque deux broches du VNC1L sont basses (d'abord *PROG*, puis *RESET*). Désormais, la mémoire peut être codée via des commandes envoyées sur l'UART. Il faut néanmoins utiliser une interface logicielle de *FTDI* pour le reprogrammer par cette méthode.

VI. DMA Controller 1 and 2 :

Les deux contrôleurs *DMA* offrent de très bonnes performances en envoyant les données depuis les deux *SIE Controllers*, *l'UART*, le bus *SPI* ou le bus *FIFO* peuvent être envoyées à la *SRAM* avec un minimum d'interventions du *MCU*.

VII. DATA SRAM :

Zone mémoire de 4ko, servant pour les variables du *MCU*, mais aussi accessible directement par les *DMAC*

VIII. NPU (Numeric Co-Processor) :

Coprocesseur permettant de déléguer des tâches arithmétiques sous 32 bits, pour le calcul, par exemple, des *FAT 32* (*un rappel sur les FAT est disponible dans la section adéquate*).

IX. UART Prescaler :

Fourni un *clock* pour *l'UART*, en faisant varier les pré-diviseurs internes, permettant de gérer des vitesses de transmission allant de 300 *baud* à 1 M *Baud*.

X. System Timer

Fourni une interruption régulière au *VNC1L*

XI. Vinculum MCU Core :

Processeur basé sur une architecture verrouillée de *FTDI*. Il est basé sur une architecture *HARVARD*. Il permet l'interfaçage d'une mémoire programme de 64ko, et d'une mémoire data d'également 64ko.

XII. E-FLASH Program ROM :

Le *VNC1L* possède 64ko de *ROM* embarquée de type Flash (*embedded Flash : E Flash*). Pour programmer cette mémoire, aucune tension spéciale n'est nécessaire.

XIII. Bootstrap loader ROM :

Petit bloc mémoire *Hardcodé*, de 512 octets, lorsque la mémoire principale (*E-Flash*) est contournée (*bypass*) lorsque PROG est mis bas, et permettant de programmer l'E-Flash via l'UART.

XIV. UART and FIFO Logic :

Interfaces séries et parallèles optionnelles, remplaçant un FT232 ou FT245.

XV. GPIO Blocks :

Broches d'entrées sorties à des fins générales. Toutes ces broches ne sont néanmoins pas accessibles à l'utilisateur. Consulter le *datasheet* en annexes.

Les firmwares

Le *VDIP2* est donc basé sur le *FTDI VINCULUM VNC1L-1A*, microcontrôleur 8bits. Selon le *fimware* choisi, l'utilisation de ce module sera différente :

Device Type	BOMS		FTDI Peripheral		HID		Printer		Hub		PC Host		Monitor		CDC	
USB Port	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
VDAP		Y	Y	Y	Y	Y	Y	Y	Y	Y					Y	Y
VDIF		Y		Y		Y		Y					Y			
VMSC		Y		Y		Y		Y		Y						Y
VDPS		Y		Y		Y		Y		Y	Y					
VCDC		Y		Y		Y		Y		Y					Y	Y
VDFC	Y	Y														

Tableau 1.3 : Fonctionnalités des différents firmware du VDIP2

D'origine, le *VDIP2* est fourni avec un *firmware* de type *VDAP*, ne permettant pas l'émulation d'un port *USB* pour le port série (*PC HOST*), néanmoins, le changement de *firmware* est très aisé, il suffit de le télécharger sur le site du fabricant (http://www.vinculum.com), et de le placer sur une clé *USB* insérée sur le port 2. Le *VNC1L* va directement *booter* sur cette dernière, et y charger le dernier *firmware* si celui-ci est antérieur ou d'un type différent à l'actuel

Les *VDIP2* commandés présentaient les versions 3.06 et 2.94 du *VDAP*, ces deux *firmware* avaient besoin d'un contrôle de flux extérieur. La version 3.68 n'en requiert pas, facilitant la manipulation. Nous avons téléchargé le *VDPS* 3.68.

A titre informatif, ce module permet de gérer des lecteurs MP3 directement, par *USB* (envois de commandes, gestion du flux audio), il permet aussi la gestion de deux stockage de masses (deux clés *USB*), …

L'interfaçage

Le *VDIP2* est présenté sous la forme d'un « *DIL40* », par le fabriquant *FTDI*/*Vinculum*. Le *VNC1L* est spécialement conçu pour la communication avec un microcontrôleur 8bits. Il possède deux modes de fonctionnements :

- *Mode commandes* : permet la création de fichiers sur clé *USB*
- *Mode Data :* permet l'émulation d'un port série (est transparent)

Le mode le plus intéressant, et apportant une grande innovation étant le mode commandes. Pour le piloter en mode commandes, nous devons l'interfacer avec notre *ADµC*, pour ce faire, 3 solutions s'offrent à nous :

- *Mode UART (via Rx / Tx)*
- *SPI (mode proche de l'I2C, sauf qu'un fil est dédié à RX et un autre à TX, contre un seul pour l'I2C)*
- *Parallèle FIFO*

Rem : le choix de l'interfaçage se fait via deux *jumpers* présents sur le *VDIP2* et est montré en annexes.

Nous connaissons les deux premiers modes, et savons que ceux-ci pourraient fonctionner avec l'*ADµC*, pour cette raison, nous n'avons pas investigué plus loin pour le choix de l'interfaçage.

L'interfaçage en SPI présente l'avantage de garder le port série libre en toute circonstances (peut faciliter énormément un éventuel débogage). Néanmoins, nous avons choisi d'utiliser ce *VDIP2* en l'interfaçant avec notre UART, pour la simplicité de commande (utilisations des procédures *printf();* et *scanf();* en C)

Dans le mode *commandes*, il y a encore deux autres modes pour l'envoi de commandes :

- *Short Command mode*
- *Extended Command mode*

Le mode étendu présente l'avantage d'être plus lisible, en effet on envoi des mots clés utilisés sous DOS, par exemple *MKD* pour *Make Directory*, en *Short command*, cette même instruction étant « 06 ». L'avantage du *Short command* réside dans le fait qu'on reçoit ***toujours*** un nombre identique de caractères après chaque instruction, et que le nombre de réponses possible est réduit par rapport au mode étendu (facilitant les tests).

Le *firmware* répond à un jeu de commandes, et renvois des réponses. Ces réponses variant de 5 à 17 caractères en mode étendu, et fixé à 3 caractères pour le mode court. Le fait de devoir recevoir 3 et non 5-17 caractères, améliore également la vitesse de communication.

Un exemple de création de fichier en mode étendu et en mode court sera démontré dans la section software.

Nous avons choisi de garder le *VDIP2*, bien qu'il s'agisse d'un module de développement, pour les raisons suivantes :

- *Firmware* programmés d'usine : le *VNC1L* acheté seul est fournis sans *Firmware*, et il faut un outillage spécial pour le flasher la première fois

- Prix de revient : le module coûte moins de 30€ tout fait, alors que le *VNC1L* coûte près de la moitié à lui seul. Le prix d'un *PCB* (pour un ou deux exemplaires) étant d'à peu près 50€/pièce… le calcul est vite fait.

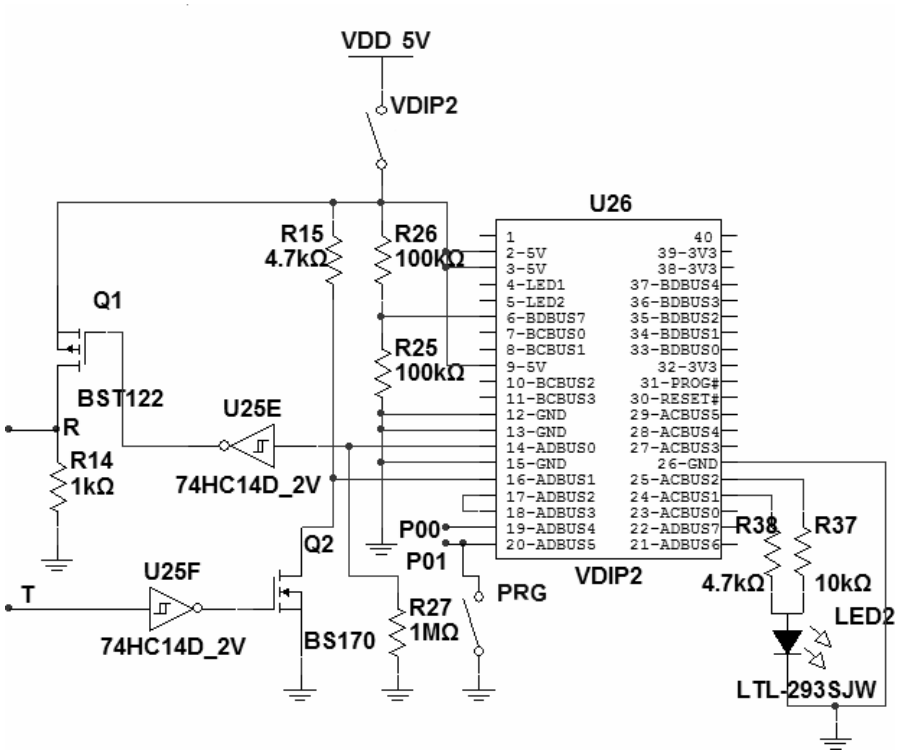


Figure 1.16 : schéma d'interfaçage du VDIP2 en mode UART

Présenté à la *figure 1.16*, l'interfaçage reste très simple.

Par choix, mais aussi par contrainte, nous avons décidé de couper l'alimentation du *VDIP2* lorsque celui-ci n'était pas utilisé (boitier/capteur fermé). En effet, la possibilité de laisser une clé *USB* sur le terrain comme stockage de masse a été rejetée au début du design du projet. De plus, une erreur dans le *firmware* du VNC1L ne permet pas de créer plusieurs fois de suite un fichier lorsque celui-ci est passé en mode *power-down*. Bug confirmé par le support de *FTDI*. Pas de solution à ce jour.

Nous avons placé un interrupteur enclenché à l'ouverture du capteur, mettant sous tension le *VDIP2*. Nous avons du isoler les broches RX et TX de notre *ADµC*, afin de ne pas alimenter continuellement ce dernier par celles-ci.

On remarquera également un interrupteur (en fait, un *jump*), sur la broche *ADBUS5* vers la masse, celui-ci permet la mise en mode « *data* », permettant la programmation par l'*USB* de notre *ADµC*.

Les deux 74HC14 présentes pour l'isolation de RX et TX sont des inverseurs à entrée *trigger de schmitt*. De simples inverseurs auraient suffis, mais nous en avions déjà placés pour les sources d'interruptions, et nous en avions encore.

Le *VDIP2* possède également des ports d'entrées sorties utilisable, et commandés en mode commandes. Nous en avons profité pour commander une *LED* bicolore indiquant l'activité d'écriture sur la clé *USB*. Dans notre configuration, ces ports ne sont utilisables que lorsque nous ouvrons le capteur.

Le format FAT 16

Notre *VDIP2* gère les FAT 12, 16 et 32. Néanmoins, le format de formatage par Windows des *flashdrive* (stockage sur de la mémoire flash, dit « clés USB ») ayant une capacité inférieure ou égale à 4 Go par défaut sont les FAT16 (même sous *Seven*). De ce fait, nous allons plutôt nous attarder sur ce dernier format. Le format FAT12 est tout bonnement obsolète.

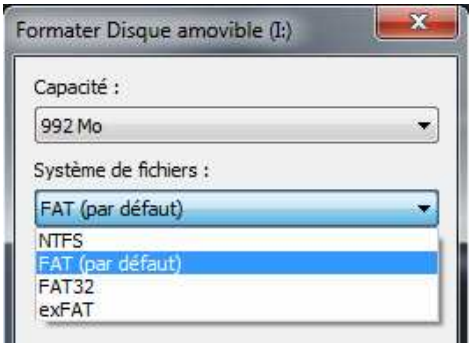


Figure 1.17 : formatage d'une clé USB de 1Go sous Window Seven

FAT signifie : *File Allocation Table*, c'est un système de fichiers qui indexe les fichiers, utilisant des adresses d'unités d'allocation sur le disque codées sur 16 bits (65536 possibilités, en théorie, 65519 voir moins dans la pratique).

Le *cluster* est l'unité d'allocation aux fichiers, et est composé de plusieurs *secteurs*. Si un *cluster* est composé de 8192 octets (8 secteurs de 1ko par exemple), et qu'un fichier pèse 9000 octets par exemple, deux *clusters* seront utilisés, le deuxième ne sera rempli que de 808 octets.

En FAT 16, la taille maximale des partitions est de 2Go, et, exceptionnellement, de 4Go. Ceci dépend de la taille du *cluster* (= taille minimale allouable), par exemple, pour une partition FAT formatée avec des *clusters* de 16ko chacun, tous les fichiers présents sur cette dernière occupera sur le volume physique un multiple entier de ce *cluster*.

En pratique, un fichier de 1ko (1024 octets), sur ce volume utilisera 16 ko d'espace. La taille des *clusters* détermine la taille totale admissible, si on a 1ko par *clusters*, en *FAT16*, on a

théoriquement un maximum de 65536 clusters allouable, soit 65536 x 1ko, ce qui fait 63.99 Mo allouable, en effet la *FAT16* ne peut plus allouer l'espace restant. Or, si on admet des *clusters* de 16ko, on pourra allouer un espace total correspondant à 65536 *clusters* (en théorie), soit 1023.98 Mo. A la ***figure 1.17***, on peut apercevoir que *Windows Seven* veut formater ma « clé *USB* » de 1Go avec 992 Mo allouables, soit 63488 *clusters* allouable réellement.

D'après la documentation (voir bibliographie), les valeurs annoncées ici sont typiques pour une clé de 1Go en FAT16.

Limitations des FAT16 :

- Taille maximale d'un fichier : 4Go – 1 octet (soit 4 294 967 295 Octets)
- Nombre maximal de fichiers par partitions : 65 524
- La racine du disque est limitée à 512 entrées (où une entrée est un fichier ou un dossier)
- Plages de dates : 1er janvier 1980 au 31 décembre 2107
- Taille maximale des noms de fichier : 8.3 (huit caractères pour le nom, et 3 pour l'extension)
- Caractères admis : tous les ASCII sur 8 bits, sauf : /\ : * ? « » <> |

Evolution FAT16 : *VFAT* qui permet entre autre de gérer des noms plus longs de fichiers/dossiers, depuis Microsoft *Windows 95*, ce mode s'applique au système de fichiers *FAT*12, *FAT16* et *FAT32*. Depuis *Microsoft Windows Vista SP1*, il existe un nouveau système de fichiers *FAT* appelé *exFAT*, on le surnomme à tort *FAT64*. Il est incompatible avec les autres systèmes de fichiers *FAT*. Il peut seulement être utilisé sur de la mémoire *flash*.

Remarque : les volumes de 4 Go formatés en FAT16 peuvent se montrer illisibles sous certains OS. Il s'agit d'un mode de compatibilité restreint.

Structure d'une FAT 16 :

En tout premier lieu, nous trouvons le *secteur de boot*, celui-ci est crée par Windows lors du formatage du volume. Lors de l'écriture sur la clé *USB*, il est intéressant de lire ce *secteur* afin de connaître le format utilisé et les attributs mémoires.

Les éléments formant le *secteur* de boot :

Position (octets)	Taille (octets)	Description
0	3	Saut vers un programme qui va charger le système d'exploitation
3	8	Nom du programme qui a formaté le disque (« MSWIN4.1 » par exemple).
11	2	Nombre d'octets par secteur (512, 1 024, 2 048 ou 4 096).
13	1	Nombre de secteurs par *cluster* (1, 2, 4, 8, 16, 32, 64 ou 128).
14	2	Nombre de secteurs réservés en comptant le secteur de *boot* (32 par défaut pour FAT32, 1 par défaut pour FAT12/16).
16	1	Nombre de FATs sur le disque (2 par défaut)
17	2	Taille du répertoire racine en nombre d'entrées (0 par défaut pour FAT32).
19	2	Nombre total de secteurs 16-bit (0 par défaut pour FAT32).
21	1	Type de disque (0xF8 pour les disques durs, 0xF0 pour les disquettes).
22	2	Taille d'une FAT en secteurs (0 par défaut pour FAT32).
24	2	Nombre de secteurs par piste.
26	2	Nombre de têtes.
28	4	Secteurs cachés (0 par défaut si le disque n'est pas partitionné).
32	4	Nombre total de secteurs 32-bit (Contient une valeur si le nombre total de secteurs 16-bits est égal à 0)
36	1	Identifiant du disque (à partir de 0x00 pour les disques amovibles et à partir de 0x80 pour les disques fixes).
37	1	Réservé pour usage ultérieur.
38	1	Signature (0x29 par défaut).
39	4	Numéro de série du disque.
43	11	Nom du disque sur 11 caractères ('NO NAME' si pas de nom).
54	8	Type de système de fichiers (FAT, FAT12, FAT16).

Tableau 1.4 : Secteur de boot FAT 16

Ensuite, vient la table d'allocation. La FAT est donc une carte où chaque nombre représente un *cluster*. Cette table est un tableau de nombres, indexé par un numéro de *cluster*. La FAT est constituée d'une succession d'entrées de 16 bits, chacune de ces entrées est associée à un unique cluster dans la zone de données. C'est le tableau qui donne l'état de tous les *clusters*.

Il y a plusieurs états possibles pour un *cluster* :

- *Cluster vide*
- *Cluster réservé*
- *Mauvais cluster : si un secteur dans un cluster est mauvais, le cluster sera rejeté*
- *Cluster utilisé + un attribut :*
    - *Pointeur sur le cluster suivant du fichier*
    - *Dernier cluster d'un fichier*

La zone de données est comme son nom l'indique, la zone où sont écrits les fichiers.
Mais, dans un *cluster* libre, différent du premier *cluster* du fichier, sera indiqué les données suivantes : (nom = *cluster* répertoire)

Offset	Taille	Description
0x00	8	Nom du fichier (rempli par des espaces) Le premier octet peut avoir l'une des valeurs spéciales suivantes : 0x00 Entrée disponible, marque également la fin du répertoire 0x05 Le nom de fichier commence en fait par le caractère ASCII 0xE5 (valeur réservée) 0x2E Entrée '.' ou '..' 0xE5 Entrée supprimée. Les outils de recouvrement de fichiers supprimés remplacent ce caractère par un autre pour restaurer le fichier supprimé.
0x08	3	Extension (rempli par des espaces)
0x0B	1	Attributs du fichier : **Bit** / **Masque** / **Description** 0 / 0x01 / Lecture seule 1 / 0x02 / Fichier caché 2 / 0x04 / Fichier système 3 / 0x08 / Nom du volume 4 / 0x10 / Sous-répertoire 5 / 0x20 / Archive 6 / 0x40 / *Device* (utilisé en interne, jamais sur disque) 7 / 0x80 / Inutilisé La valeur d'attributs 0x0F est utilisée pour désigner un nom de fichier long.
0x0C	1	Réservé, utilisé par NT
0x0D	1	Heure de création : par unité de 10 ms (0 à 199).
0x0E	2	Heure de création : **Bits** / **Description** 15-11 / Heures (0-23) 10-5 / Minutes (0-59) 4-0 / Secondes/2 (0-29)
0x10	2	Date de création : **Bits** / **Description** 15-9 / Année - 1980 (0 = 1980, 127 = 2107) 8-5 / Mois (1 = Janvier, 12 = Décembre) 4-0 / Jour (1 - 31)
0x12	2	Date du dernier accès ; voir *offset* 0x10 pour la description.
0x14	2	Index EA (utilisé par OS/2 et NT) pour FAT12 et FAT16 ; 2 octets de poids fort du numéro du premier *cluster* pour FAT32
0x16	2	Heure de dernière modification ; voir *offset* 0x0E pour la description.
0x18	2	Date de dernière modification ; voir *offset* 0x10 pour la description.
0x1A	2	Numéro du premier *cluster* du fichier (FAT12 et FAT16) ; 2 octets de poids faible de ce numéro (FAT32).
0x1C	4	Taille du fichier

Tableau 1.5 : Cluster répertoire / catalogue

Cette zone est aussi parfois appelée le « catalogue »

Marche à suivre pour l'écriture d'un fichier en FAT 16 :

- Aller lire dans la FAT tous les clusters libres
- Coder le cluster répertoire, dans la zone de données, en donnant le premier cluster du fichier en l'adresse 0x1A
- Se rendre dans la FAT à l'entité correspondante au premier cluster du fichier, et indiquer si celui-ci est le dernier de la chaîne ou pas.
- Dans la zone des données, on code les différents clusters du fichier

Marche à suivre pour la lecture d'un fichier en FAT 16 :

- Aller chercher dans la zone des données / catalogue le cluster répertoire dont l'attribut « DIR_Name » (0x00) correspond au nom du fichier recherché
- Lorsque le cluster répertoire est trouvé, on cherche le premier cluster du fichier (0x1A)
- Aller dans la FAT à l'entité correspondante au premier cluster du fichier, et regarder s'il s'agit du dernier cluster formant le fichier. Dans la négative, aller au deuxième et répéter cette étape.
- Lorsqu'il s'agit du dernier cluster, on a la chaîne de clusters permettant d'aller lire le fichier dans les zones de données pointées par tous ces clusters.

Il faut donc avoir recours au service du catalogue et de la FAT pour pouvoir lire l'intégralité des données contenues dans un fichier donné.

Circuit imprimé - Principal

Nous avons tracé le circuit imprimé sur *Ultiboard*, après avoir dessiné le schéma complet dans *Multisim*. *Rem : nous n'avons pas simulé le circuit dans Multisim pour la bonne et simple raison que plusieurs de nos circuits n'existaient pas, et que nous ne voulions pas perdre de temps pour encoder son fonctionnement.*

L'avantage de faire le schéma de principe dans *Multisim* est la génération automatique d'une *netlist* lors de l'exportation dans *Ultiboard*. Cette *netlist* est importante puisqu'elle contient toutes les connexions virtuelles entre les différents composants. *Ultiboard* permet de nous dire directement quels sont les éléments à interconnecter entre eux, et nous prévient lorsqu'il manque une connexion ou qu'une connexion est mal faite.

La licence de cette suite de *National Instrument*, nous a été prêtée par M. Morren, gérant de *GmE*.

Nous avons réalisé un circuit imprimé double face, dont voici le récapitulatif, fourni par U*ltiboard* :

Total number of pins:	*569*
Pins in a net:	*506*
Not connected pins:	*63*
Test pins:	*0*
Jumpers:	*0*
Total number of vias:	*156*
Total number of connections:	*388*
Unrouted connections:	*0*
Completion:	*100%*
Total number of components:	*140*
Total number of nets:	*118*

Nous avons désiré placer les bascules et les inverseurs présentés plus haut, en dessous du « *DIP40* » formé par la *VDIP2*, pour une meilleure intégration des composants.

Nous avons également jugé utile d'y insérer un deuxième *IO Expander*, permettant une évolution du projet, et de placer deux *RAM* Non volatile.

Etant donné qu'il s'agit d'un prototype, nous avons connecté le plus possible de broches de l'*ADμC* 832 à des *PIN HEADER*, facilitant un éventuel débogage.

Notre carte mesure 109mm sur 78 mm et possède deux plans de masse.

Les deux Layers, la sérigraphie, la vue 3D et les photos du travail fini du *PCB* sont disponibles dans la section annexe.

Soudage du PCB

1) Tests préliminaires

a. Vérification de l'isolement entre *GND* et *VCC*
b. Vérification de la continuité entre toutes les masses
c. Vérification de la continuité entre tous les points positifs

2) *Soudage de L'alimentation 5V*

Pont de diodes fonctionnel : 12.8V en entrée, 11.9V en sortie = 0.9V « perdus » aux bornes du pont, lorsque peu de courant traverse celui-ci !
Les diodes de sélections de la source (Batterie ou pile de secours) : Ok : 0.548V

➔ Tension disponible au régulateur 5V : $Vin - 0.9V - 0.548V$.

Alimentation 5V : sérigraphie incorrecte de la diode du régulateur à découpage : sans conséquences.
Tests avec 3V en entrée : pas de mise en sécurité.
12V en entrée : régulation de 4.92V, pour 297µA Absorbé.

3) Alimentation 6V

6,00 V en sortie, le ON-OFF ayant l'effet désiré.

Consommation les deux régulateurs ensemble : 6.04mA. D'après le *Datasheet*, le LM2575 (6V) devrait consommer entre 5 et 9mA : Ok

4) Soudage de l'ADµC

Lorsque celui-ci était soudé, l'alimentation 5V s'est mise en sécurité. Après investigations, il semblerait que nous avions inversé *VCC* et *GND* sur un des 4 « côtés » de l'*ADµC*. Les 4 *GND* et *VCC* étant reliés ensemble de manière interne : court-circuit.

Nous avons dû donc couper les pistes, et tirer du fil émaillé pour reconstituer le 5V où il devrait l'être.

Consommation à 2 MHz (défaut), et les deux régulateurs allumés : 11.80 mA : OK

5) Circuits en dessous du VDIP2

PCF8574 : IO Expander
Bascules D et inverseurs : tests concluant (mettre un état sur une entrée de l'inverseur et vérifier s'il est bien inversé en sortie, et vérifier la mémorisation, ainsi que le *Set et Reset* d'une bascule)

Rem : nous avons oublié les deux résistances de *pull up* pour *SDA* et *SCL* du bus *I2C.*

6) VDIP2

Après avoir branché sur son support DIL40 le *VDIP2*, nous avons essayé d'établir une communication entre le PC et l'*ADµC*. Pour ce faire, il a fallu souder les transistors Q1 et

Q2. Nous voyons bien sur un terminal le message cohérent qu'envoie l'*ADµC* lors d'un « *debug – reset* » :

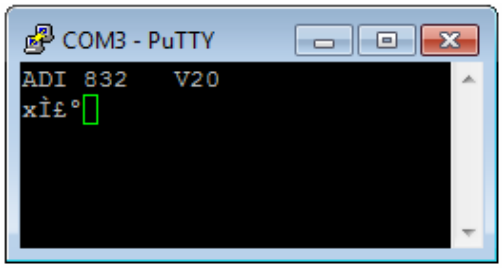


Figure 1.18 : Communication à sens unique réussie

Un chargement de programme n'est pas fructueux… Après analyses :

- Le transistor Q2 est monté à l'envers : erreur dans la sérigraphie.
- Après changement du transistor, la tension sur la broche RX de l'*ADµC* varie de 5 à 2.5V, et non de 5 à 0V.

Comme on peut le voir à la *figure 1.19*,

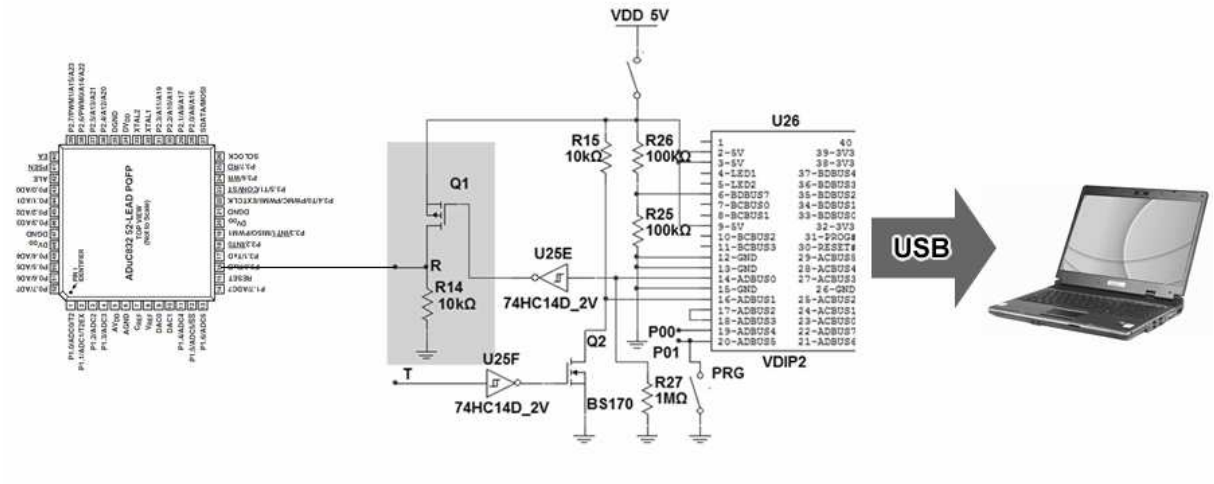


Figure 1.19 : Interfaçage du VDIP2 en mode UART

La tension en sortie de l'inverseur 74HC14 était cohérente (5V-0V). Sachant que R14 est connecté directement à RX de l'*ADµC* (*net* nommé R dans *Multisim*), port possédant une résistance de *pull-up* interne, de 10kΩ.

Lorsqu'un état bas veut être transmis par le PC, nous avons le PMOS qui ne conduit pas (« 0 » inversé). Nous avons normalement, le potentiel bas tiré par R14. Mais comme ces résistances sont identiques, nous sommes juste en présence d'un pont diviseur de tension, et nous sommes dans une zone indéterminée, point de vue niveau logique. En remplaçant R14 par une résistance de 1KΩ, nous obtenons une tension variant de 5V à 0.46 V, tension perçue comme un « 0 » logique par l'*ADµC*.

L'inconvénient est que nous consommons plus que prévu.

Test : chargement d'un programme : OK !

7) Soudage des RAM et du TLC556.

Après avoir placé le TLC556 et les composants connexes, nous voyons bien qu'il oscille : Ok

RAM : les *shapes* pris sur *Ultiboard* ne sont pas corrects ! L'empattement réel est plus important que prévu. Néanmoins, c'est soudable !
Après avoir soudé les résistances de *pull up* sur *SDA* et *SCL*, l'écriture et la lecture d'un octet sur les deux RAM est ok.
Nous avons également profité pour tester l'IO Expander PCF8574 : Ok

8) Horloge I2C

Elle compte bien avec une fréquence proche de l'Hertz : OK.

Tout le *PCB* est utilisable à l'heure actuelle !

Relevé des consommations

Nous avons dans un premier temps déterminé le meilleur compromis vitesse d'exécution par rapport à la consommation, de notre carte à base de l'*ADμC* 832, en mode « normal », sans *VDIP2*, chauffage ou moteurs…

Fréquence	**Consommation**
500kHz	6.80 mA
2MHz	9.50 mA
16MHz	21.05mA

Tableau 1.6 : Consommation de notre carte selon la fréquence

Cette consommation comprend tous nos circuits intégrés soudés sur notre *PCB*. Néanmoins, à 500kHz, le *VDIP2* ne fonctionne pas, et à 16MHz, notre librairie pour l'I2C ne fonctionne plus (revoir les délais). Nous avons choisi 2MHz, valeur par défaut. Néanmoins, avec l'interface utilisateur en C#, une fréquence de 2MHz ne permet pas la communication correcte. Nous avons décidé de modifier la fréquence à 8Mhz lorsque l'on passe en mode *DATA*, et repassons à 2MHz ensuite.

Afin de réduire au minimum notre consommation, nous allons mettre l'*ADμC* en *idle mode*, abaissant la consommation (à 2MHz) de 9.50 mA à 7.40 mA. (10.4 mA lors d'acquisitions)

Consommation du VDIP2 :

Usage	**Consommation**
Power Down	2.10 mA
Normal mode	23.2 mA
Ecriture sur clé	115 à 130 mA

Tableau 1.7 : Consommation du VDIP2

Mais, comme expliqué avant, le *Power Down* possède un problème, et nous avons donc oublié ce mode.

Notre Alimentation 6V consomme 5.90 mA en fonctionnement, 86 µA en *Power Down*.

Consommation des Servomoteurs

Nous avons deux Servomoteurs, un pour le couvercle (3Nm : le *Futaba* S9157), et un pour le carrousel (0.3Nm, le *Futaba* S3003). Ces deux moteurs sont similaires par l'empattement, mais possède une consommation totalement différente, en toutes circonstances.

Mode	S3003	S9157	Remarque
Au repos	9.8mA	10.8 mA	Sans impulsions
Sans forcer	100 à 300 mA	0.5 à 1A	Selon l'angle
En forçant	625 mA	2.07A	Couple MAX

Tableau 1.8 : Consommation de nos servomoteurs

Consommation du chauffage

Tension	Courant
11V	220.0mA
11.5V	230.0mA
12V	240.4mA
12.5V	250.30mA
13V	259.5mA
13.5V	269.5mA

Tableau 1.9 : Consommation du chauffage en fonction de la tension de service

La consommation est un peu plus élevée que prévue par le fabriquant, néanmoins, l'évacuation de la rosée est très rapide, ce qui nous permet de chauffer peu.

Circuit imprimé : Barrière infrarouge

Vu le coût de fabrication d'un *PCB* (environs 100€ TVAC, pour une série de 3), nous avons décidé de le réaliser nous-mêmes.

Les vues 3D et le *layer* sont présent dans la partie annexe.

Aucun problème rencontré avec ce *PCB*.

Problèmes rencontrés

(*En plus des problèmes cités dans le soudage du PCB*)

Sur le *PCB* principal, le plus gros problème se situe au point de vue de l'alimentation 6V, en effet la bobine pour le régulateur à découpage choisi était une bobine de *choke*, et non une bobine prévue pour l'emmagasinage d'énergie. Pour dessouder la bobine, nous avons abîmé une des deux pastilles où celle-ci était soudée. Une tentative de réparation a été entreprise, mais après tests, notre alimentation ne pouvait pas débiter un très gros courant dû aux résistances de contact. Nous avons été contraints de solutionner le problème avec une alimentation externe. Par manque de temps nous avons utilisé le régulateur que nous avions sur place : le *LM317* (linéaire). Après mesure, le courant débité est de 1.2A pour 5.8V, ce qui est tout à fait acceptable pour nos Servomoteurs.

Néanmoins, afin de valider le design, nous avons réalisé sur un autre *PCB* la même alimentation, avec une bobine appropriée. Un courant de 1.2A pour 5.8V est également mesuré.

Figure 1.20 : Réalisation de l'alimentation 6V régulée sur une deuxième carte

Nous avons également remarqué que la qualité des *PCB* fabriqués par *Eurocircuits* est un peu en deçà de nos attentes. En effet, le fait de dessouder une ou deux fois au même endroit fait sauter le trou métallisé et sa pastille. Dans une prochaine version nous agrandirons la surface de cuivre de la pastille (malgré que nous soyons tout à fait dans les normes).

Nous avons placé sur le *PCB* des *PIN HEADER* à verrouillage pour tous les périphériques, y compris pour la barrière infrarouge et l'encodeur gray. Cependant la disposition de celui-ci n'est pas parfaitement ergonomique.

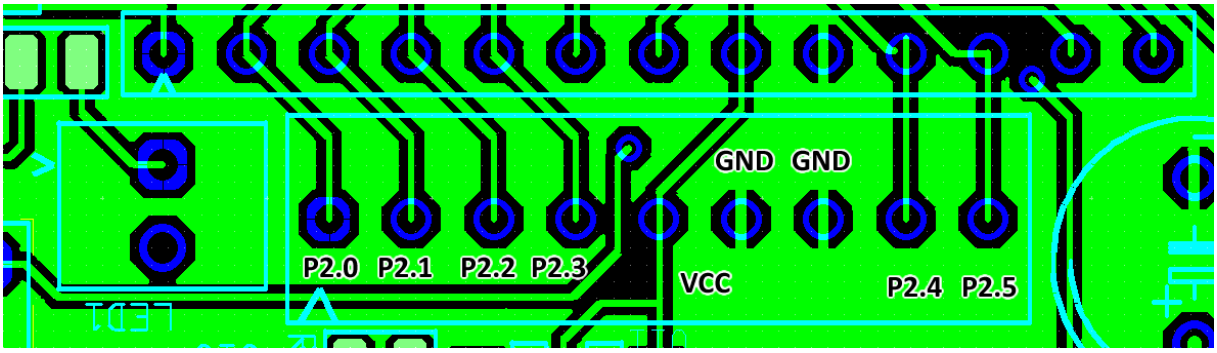


Figure 1.21 : Zoom sur la zone du pin header à verrouillage pour la délocalisation de l'encodeur GRAY et de la barrière infrarouge

Comme on peut le voir à la *figure 1.21,* nous avons placé l'encodeur *GRAY* sur P2.0 à P2.3 (+ *GND*). La barrière infrarouge ayant besoin de : deux fois *GND* et deux fois *VCC* (un pour le *PCB* de l'émetteur, et un pour le *PCB* du récepteur), en plus des signaux de contrôle : P2.4 (= état de la *LED*), et P2.5 (= état du phototransistor)

Il aurait été mieux de placer comme ceci :

P2.0 – P2.1 – P2.2 – P2.3 – GND – VCC – GND - P2.4 – VCC – GND – P2.5

Afin de placer les alimentations proches et au bon nombre des périphériques, évitant de devoir croiser des câbles.

Un autre problème détecté, est que l'on ne passe pas par le pont de *Graëtz* avec la pile de secours, une inversion est toujours possible.

Et en dernier, la *RTC* qui nous donne un top assez loin de l'Hertz : en effet, sans souder le condensateur visible sur le schéma de principe, nous avons relevé une période de 977.9492 ms, au lieu et place de 1000 ms, ce qui est très mauvais, en effet :

$1000 - 977,9492 = 22,0508ms$ de décalage à chaque secondes,

Sur une heure,
$22,0508.3600 = 79382,88ms$ soit 1min 19 secondes et 382,88 ms de décalage

Sur une journée
$79,38288.24 = 1905,18912s$ soit 31minutes, 45 secondes et 189,12 ms de décalage

Sur une année de 365 jours :
$1905,18912.365 = 695394,029s$ soit 8jours, 1heure, 9minutes, 45 secondes et 28,8 ms de décalage.

Ce décalage supérieur à 8 jours par an est énorme. Nous avons soudé un condensateur de 22pF, et obtenu une période de 999.6ms. Ce qui correspond sur un an à un décalage de 3heures, 30 minutes, 14 secondes, 40 millisecondes.

Ce résultat est tout à fait acceptable.

Rem : cette mesure a été faite à une température ambiante de 22°C.

Récapitulatif des éléments rendant notre projet défensif point de vue Hardware

- Pont de diodes pour la batterie
- Filtre anti-rebond pour les sources d'interruptions
- Ram non Volatile pour le projet
- Pile de secours
- Détrompeur pour les connecteurs à verrouillage (*voir figure 1.22*)
- Barrière infrarouge pour le positionnement optimal
- Deux emplacements possibles pour le capteur de pluie
- Utilisation de bascules pour les sources d'interruptions, afin d'être sûr de n'en rater aucune.
- Résistance de limitation de courant au régulateur 5V
- Piste dont la largeur est diminuée en un endroit précis, si trop de courant est absorbé par le projet, limitant les zones de pannes.

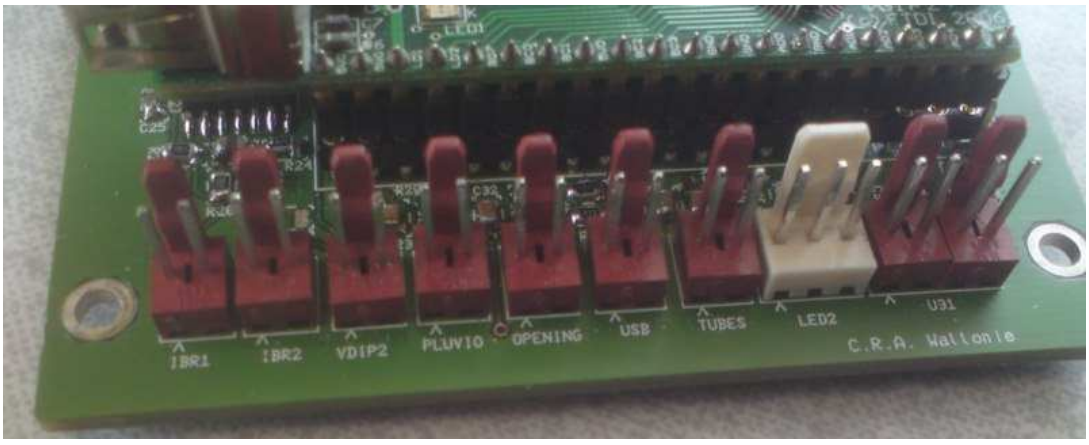


Figure 1.22 : Pin header à verrouillage avec détrompeurs

2. Programmation

Nous avons programmé notre *ADµC 832* en C, sous le compilateur *KEIL*.

Je vais dans un premier temps décrire dans cette section comment fonctionne le programme, ensuite je vais établir une fiche détaillée de chaque fonction qui constituera mon analyse modulaire (ce que chaque fonction fait, les arguments entrée et sortie, …) et enfin, je vais expliquer brièvement la programmation du module *VDIP2*.

I. Analyse du programme

Programmation embarquée

Les organigrammes sont disponibles dans la section 2 de ce chapitre : « Description modulaire ».

Notre hardware est fait de manière à n'avoir que des évènements, ou interruptions, soit extérieurs (boutons poussoirs, pluviomètre, *RTC*, Port série,…), soit internes (*TIC* : *Time Interval Counter*).

Il devra donc y avoir un système permettant de discriminer quelle interruption a demandé d'exécuter son code. Discriminer la *TIC* des interruptions extérieures est natif, puisque les vecteurs d'interruption prévus dans *l'ADµC* sont différents, et que en conséquent, par programmation, on peut écrire deux services différents.

Toutes les autres interruptions (exceptée celle du port série), sont sur la même broche d'interruption : *int0*. Pour pouvoir dissocier quelle est la source physique qui demande l'interruption, il faut répliquer chaque source sur un port d'entrées, ce port d'entrée est chez nous la *PCF8574 IO Expander* de Philips, se commandant par I2C. Comme expliqué dans la partie électronique, des bascules sont prévues afin de ne pas perdre une interruption, puisque l'I2C est relativement lent (système d'adressage, …), par rapport à la lecture d'un des ports « direct » de *l'ADµC*.

Une fonction devra aller vérifier un à un l'état de chaque bascule, répliquée sur notre *IO Expander*, et positionner un *flag* à l'état haut, permettant à une autre fonction de lancer le service adéquat. L'heure de chaque demande d'interruption est alors sauvée dans des variables.

Faire de cette sorte permettra de ne rater aucune interruption, en effet, si deux arrivent en même temps (ou du moins juste après le temps nécessaire à la lecture par l'I2C de notre *IO Expander*), deux *flags* seront mis à 1, qui permettra à la fonction adéquate de lancer les bons services d'interruptions.

Si nous n'avions pas fait comme ceci, et que les interruptions 1 et 3 sont demandées en même temps :

- Vérifier Interruption 1 : OUI
- Sauver l'heure de l'interruption 1
- Lancer l'interruption 1
- Vérifier interruption 2 : NON
- Vérifier l'interruption 3 : OUI
- Sauver l'heure de l'interruption 3
- Lancer l'interruption 3
- …

Ce procédé aurait eu plusieurs défauts :
1) Le temps d'exécuter l'interruption 1, la broche int0 est bloqué, d'autres interruptions pourraient ne pas être détectées
2) L'horodatage de l'interruption 3 serait l'horodatage lors de son lancement, et non de sa demande (temps d'exécution de l'interruption 1)

Il y a aussi un autre problème, on ne peut interrompre un périphérique en train de communiquer sur le bus I2C, pour lancer une autre application qui l'utilise, **ce qui implique de bloquer toutes interruptions lorsque l'I2C est utilisé.**

Nous allons vérifier au pas à pas chaque source, positionner les *flags* éventuels, et réarmer les bascules.

Une autre fonction va se charger de lancer les fonctions adéquates.

Une interruption synchrone, la *RTC*, qui toute les 10 secondes va demander à *l'ADµC* de faire des acquisitions de température et du taux d'humidité relative, via le capteur *sensirion*. On va devoir vérifier combien d'acquisitions ont été faites sur l'heure en cours, et comparer au nombre d'acquisitions qui auraient dû être fait, permettant de dissocier l'installation, avec remise à l'heure par le PC, à par exemple 12 :59 :40, cela signifierait qu'il ne reste plus beaucoup d'acquisitions à faire dans l'heure, malgré qu'on en ait encore fait aucune (important pour effectuer la moyenne, en fin de l'heure).

A minuit, il faudra tourner d'une position le carrousel, si le pas de quantification est d'un jour. S'il est de 2 jours par exemple, il faudra tourner tous les deux jours, ceci implique la création d'une variable locale (*static*), déterminant s'il faut tourner ou pas (décrémentée chaque jours, lorsqu'elle vaut zéro, il faudra tourner).

Il nous faut cadencer les mesures de fréquences pour décider de l'ouverture du couvercle, mais aussi de minuter le timeout du pluviomètre. Etant donné qu'on ne fera plus d'acquisitions de fréquence lorsque le couvercle est ouvert, on peut se permettre d'employer la même ressource physique : la *TIC* (*Time Interval Counter, pouvant compter de 1/128ème de secondes à 255 heures*), celle-ci va nous permettre de :

1. Cadencer les mesures de fréquences à 1/128ème de secondes (via les compteurs T0 et T1 16bits)

Pendant ces mesures, toujours vérifier que le capteur est toujours armé :

   a. Température du capteur proche du point de rosée ? Si oui chauffer ($\Delta t° \geq 3.5°C$)
   b. Température du capteur > 70°C : arrêter de chauffer
   c. Fréquence < 11520 Hz : Chauffer
   d. Fréquence > 12928 Hz : ne pas chauffer
   e. A la fermeture du couvercle (timeout) chauffer tant que fréquence < 12928Hz

2. Timeout : chaque basculement relancera la *TIC* d'une valeur paramétrée (en minutes). Par défaut, 15 minutes.

Attention, il faut à chaque fois vérifier l'état de santé de la batterie avant d'entamer une action consommatrice (moteur, chauffage)

On va mesurer la température du capteur via une *CTN*, monté dans un pont (15kΩ + CTN 10kΩ @25°C), via l'équation de *Steinhart-Hart*, dont le calcul des coefficients est expliqué sur *Wikipédia* (voir bibliographie). La mesure se fait via *l'ADC*, en mode 10 bits. La résistance de référence est de 15kΩ permettant une dynamique plus importante dans les températures possibles en Belgique (de -20°C à +40°C), comme le montre la courbe expérimentale :

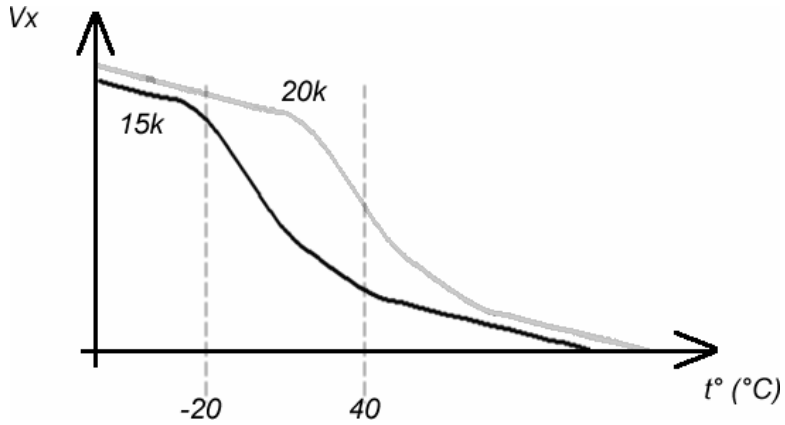


Figure 2.1 : courbe liant la dynamique de la tension à mesurée, par rapport à la résistance de référence

Lorsque le technicien viendra changer les filtres, et récupérer les données, il faudra arrêter les acquisitions synchrones (cadencées par la *RTC*), ainsi que les acquisitions de fréquence (cadencées par la *TIC*). L'ouverture de la boîte déclenchera une interruption, et mettra en route le module *VDIP2*. Le programme de l'ouverture de la boîte sera alors chargé, celui-ci s'occupera de l'initialisation du *VDIP2*, et des *LED* du panel. Ensuite, le programme devra attendre une interruption extérieure au niveau du panel :

- Bouton *USB* permettant de :
  - Charger les données sur la clé *USB* si celle-ci est présente
  - Initier la communication *PC* ⇔ *ADμC*, si aucune clé n'est détectée.

- Bouton filtres changés, indiquant à *l'ADμC* que ceux-ci ont été changés.

Lorsque le technicien le demandera, nous allons créer un fichier tabulé, exploitable sous Microsoft Excel, de manière très simple, en mettant en forme avec des *TAB* et des *CR*. On va créer un fichier *data* et *log* par semaine. On devra donc à ce moment là aller chercher les variables présentes dans la *FRAM1* et *FRAM2*.

Le paramétrage par le PC requérant une interface graphique simple d'utilisation (cahier des charges), nous avons choisi de le faire en C#, puisque nous avons déjà abordé ce langage au cours de programmation de M. Morren. Nous avons remarqué qu'il fallait absolument travailler à une vitesse supérieure de 2 MHz pour établir la connexion entre l'*ADµC* et le *PC*.

Nous plaçons après l'appui par le technicien sur le bouton « *USB* », sans clés *USB* présente, *l'ADµC* à 8MHz, car à 16MHz, les délais calculés via les librairies que nous avons pour *l'I2C* ne sont plus cohérents, et donc la transmission est mauvaise.

Il faut également pouvoir dissocier s'il s'agit du premier reset, ou d'un reset effectué après (changement de batterie sans pile de secours, …) pour ce faire, nous allons placer dans *l'EDATA* (mémoire non volatile de *l'ADµC*), une valeur particulière (*0xAA soit 0b 1010 1010*), dans une page quelconque de l'EDATA. A l'initialisation, nous vérifions que cette valeur est présente, si elle l'est, c'est qu'il s'agit d'un deuxième reset, et allons chercher les réglages introduit par l'utilisateur (timeout, pluviomètre, ….) dans *l'EDATA*, ainsi que les offset des *FRAM*.
Pour la rotation des moteurs, nous allons utiliser les deux *PWM*, se commandant séparément. Néanmoins, l'état de repos de la PWM (et donc du moteur) non utilisé devra obligatoirement être à 1, en effet, l'état 0 est impossible avec une *PWM*, le temps que l'électronique interne compare, il y a un petit flanc, et donc une impulsion est transmise (longueur : 42µs mesurée) au moteur que l'on ne voulait pas tourner.

Le Servomoteur pour le carrousel doit être piloté à 50Hz, et celui pour le couvercle à 300Hz. C'est la seule différence qui réside entre ces deux moteurs, le reste étant identique. Nous allons utiliser la PWM en mode « *Dual 8-bits PWM »*

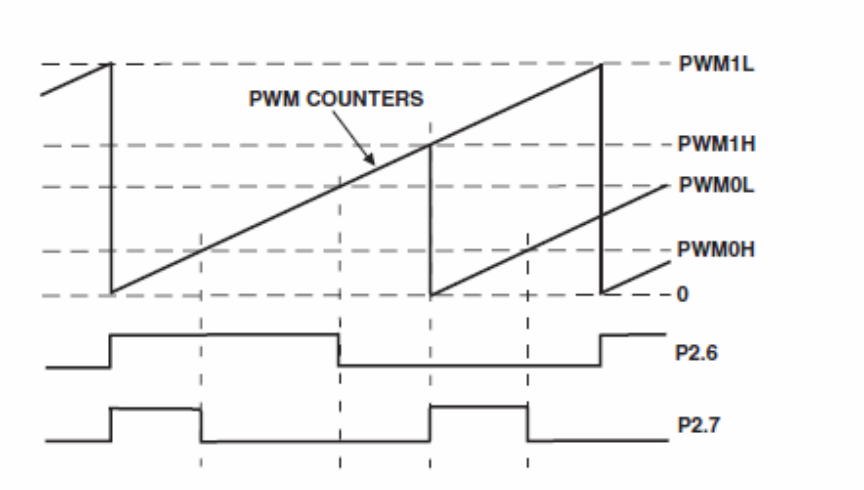


Figure 2.2 : fonctionnement en mode dual 8bit des sorties PWM

Visible sur la *figure 2.2*, le fonctionnement de la PWM en mode DUAL8 bits. Un compteur de PWM va compter jusqu'à la valeur maximale, fixée dans PWM1L. Cette valeur permettant de déterminer la fréquence des impulsions.

Ces deux valeurs seront donc différentes que l'on veuille actionner le carrousel ou le couvercle.

Pour le couvercle (300Hz), nous avons choisi comme source de cadencement du compteur, la fréquence du cristal (32,768 kHz), sans pré diviseur (*figure 2.3*). Ceci signifie donc qu'en une seconde, le compteur 16 bits (f*igure 2.2 et 2.4*) aura 32768 impulsions, et qu'il lui faudra deux secondes pour arriver à son maximum. La fréquence minimale est donc de 0.5 Hz.

Pour obtenir 300Hz en sortie, on doit obtenir à 32768 Hz un certain nombre d'impulsions au niveau du compteur, pour ce faire on détermine *PWM1L* (= le maximum du compteur, déterminant la fréquence, et quelconque pour *PWM1H*, puisque seul *PWM1L* joue sur P2.6 où est mis le servomoteur du couvercle) :

$$PWM1L = \frac{32768}{300} = 109,22 \cong 109$$

Il faudra donc 109 impulsions à 32768 Hz pour sortir du 300Hz, Lorsque le *PWM Counter* contiendra 109, si PWM0L (la valeur de la largeur d'impulsion, étant déterminée par PWM0L pour la sortie P2.6) vaut par exemple 55, nous aurons donc un rapport cyclique proche de 50%, à 109 dans le compteur, P2.6 repassera haut.

CDIV1
CDIV0

PWM Clock Divider
Scale the clock source for the PWM counter as shown below:

CDIV1	CDIV0	Description
0	0	PWM Counter = Selected Clock /1
0	1	PWM Counter = Selected Clock /4
1	0	PWM Counter = Selected Clock /16
1	1	PWM Counter = Selected Clock /64

CSEL1
CSEL0

PWM Clock Divider
Select the clock source for the PWM as shown below:

CSEL1	CSEL0	Description
0	0	PWM Clock = $f_{XTAL}/15$
0	1	PWM Clock = f_{XTAL}
1	0	PWM Clock = External input at P3.4/T0
1	1	PWM Clock = f_{VCO} = 16.777216 MHz

Figure 2.3 : tableau du cadencement du compteur de PWM

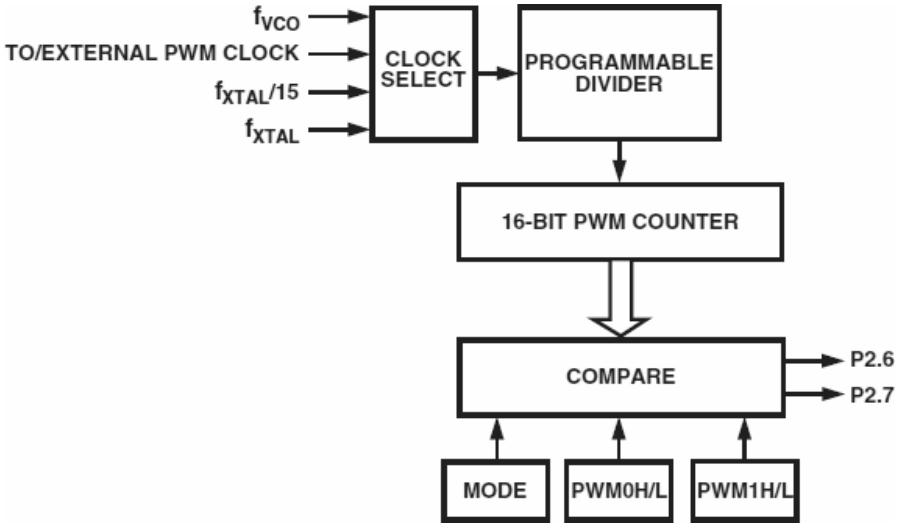


Figure 2.4 : Diagramme fonctionnel des PWM

Calculs similaires pour le carrousel, à 50Hz, nous avons préféré prendre un diviseur/4 de la fréquence source du compteur (***figure 2.3)***, pour obtenir une plus grande dynamique.

Le compteur 16Bits des PWM (***figure 2.4***) va s'incrémenter à

$$f_{PWMCounter}\frac{32768}{4}=8192[Hz]$$

Il faudra donc 8 secondes pour que le compteur arrive à 65535.
Pour obtenir du 50Hz, nous allons positionner *PWM1H* à 164

$$PWM1H=\frac{8192}{50}=163,84\cong 164$$

Comme précédemment, il faudra 164 impulsions dans le compteur 16bits, pour obtenir du 50Hz en sortie.

Nous avons vérifié nos fréquences en sortie de nos PWM :

A) le carrousel : 50Hz – 50.94Hz

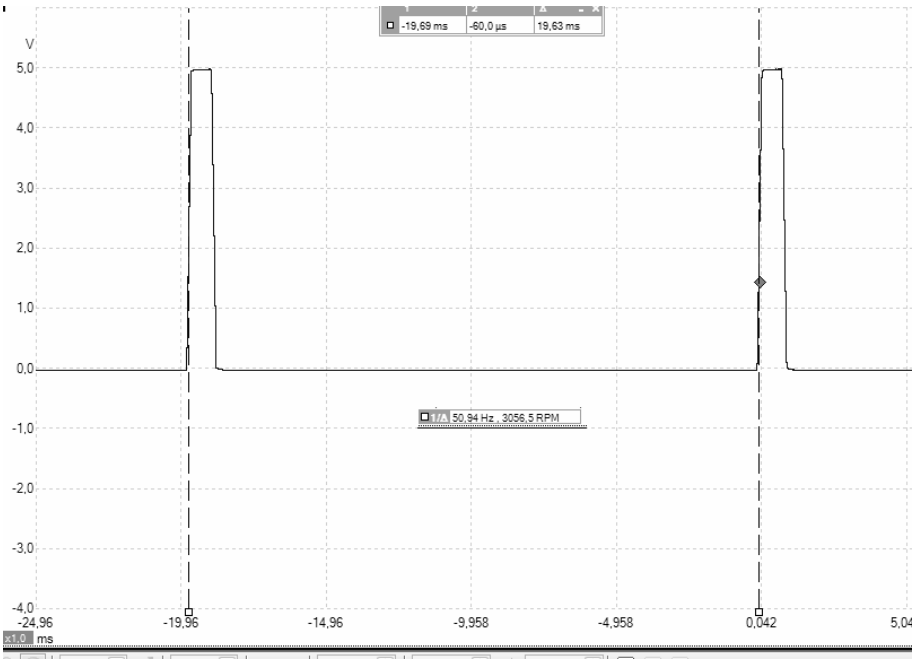


Figure 2.4 : Relevé de la fréquence du PWM commandant le carrousel (50Hz)

Visible aux *figures 2.4 et 2.5*, les relevés de fréquence de nos *PWM*, fort proches de nos calculs. Les écarts se justifient par les arrondis que nous avons faits, mais le comportement reste toutefois totalement sain et normal.

B) Le couvercle : 300Hz – 297.6Hz

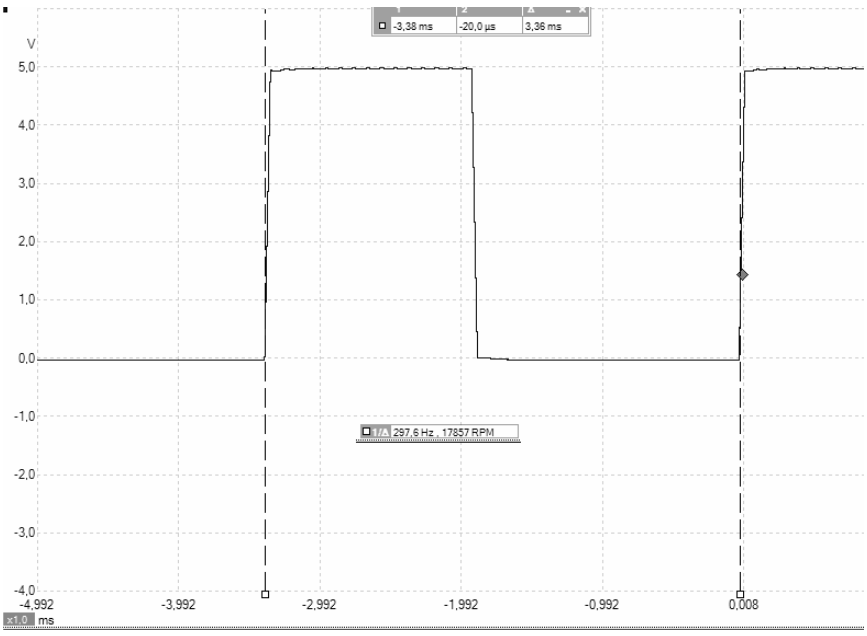


Figure 2.5 : Relevé de la fréquence du PWM commandant le couvercle (300Hz)

Listing des variables globales :

On sent intuitivement que ce programme va posséder beaucoup de variables globales, qu'il faut, en général éviter.

Bit :

Flags d'interruptions : INTRTC, INTPLUVIO, INT*USB*, INTOP
Flags d'état : Ouverture, lowbat, sensib

Char (1 octet)

Variables d'horodatage : timeRTC, timePLUVIO, timeOPEN, time*USB*, buftime, chacune de ces variables font 6 octets.

Variables du paramétrage : timeoutpluvio, auget, quantif

Buffer pour les *scanf* du *VDIP* : 60

Integer

Offset des mémoires non volatiles : offset et offset2 (FRAM1 et FRAM2)
Variable du paramétrage : pause pour l'ouverture/fermeture du couvercle (déterminant sa vitesse d'exécution)

Variable de pluviométrie : ACQpluvio : nombre de basculements

Float (4 octets)

Variables de pluviométrie : cumul et débit (instantané, minimal, maximal et moyen)

Remarque : pour placer dans notre mémoire non volatile les données de type *int* et *float*, nous utilisons des unions, puisque cette mémoire n'acceptant que des données octets par octets.

Remarque 2 : certaines de ces variables sont cependant modifiées que dans une procédure, comme par exemple *Pause, quantif, sensib*, … mais sont utilisées dans d'autres. Les variables de l'horodatages, sont pour la plus part modifiées que dans *scrutint*(); mais utilisées également ailleurs…

Avoir mis le *buffer* pour la fonction *scanf()*; appelée lorsque le *VDIP* fonctionne est en global, puisque utilisé à plusieurs endroits également, ceci permet de ne pas multiplier le nombre d'octets locaux, ces variables étant peu importantes.

Interface de paramétrage

Comme cité précédemment, nous avons codé une interface en C# permettant le paramétrage des données selon le désir de l'utilisateur.

Cette interface permet de :

1. Remettre à l'heure la *RTC*
2. Remettre à jour la *RTC*
3. Modifier le timeout
4. Modifier le pluviomètre : pluviomètre basculant pour 2mm par exemple.
5. Paramétrer le seuil de sensibilité à l'ouverture du couvercle
6. La vitesse d'ouverture & de fermeture du couvercle
7. le pas de quantification : combien de jours par tubes (1 à 7)

Comme vu précédemment, cette interface sera utilisée avec l'interruption du port série. Nous avons prévu un protocole à nous, permettant également de trouver automatiquement le port *COM Windows* sur lequel l'interface est reliée.

Bien que peu important pour des liaisons à faible longueur de câble, nous avons néanmoins fait attention à la distance de HAMMING :

La distance de Hamming c'est le nombre de bits séparant deux codes valides

En d'autre termes, si l'on veut transmettre l'ASCII de A : 01000001, mais qu'on reçoit 01000011 (C), et que ces deux codes sont valides, la distance de Hamming est de 1. En revanche, si ce code n'est pas valide, le système pourra le détecter et demander de renvoyer, par exemple.

Nos commandes :

- Trouver l'*ADµC* : F : 01000110
- Lire l'heure et la date de la *RTC* : R : 01010010
- Mettre à jour : D : 01000100
- Mettre à l'heure : T : 01010100
- Modifier la sensibilité : S : 01010011
- Modifier la vitesse d'ouverture : V : 01010110
- Modifier le timeout du pluviomètre : t : 01110100
- Modifier le pas de quantification : Q : 01010001

La distance de Hamming est suffisante (2 bits), pour la pluspart, juste entre R et S, un seul bit ne change, ce qui est toutefois bon puisque distance courte, et dispositif peu critique (on ne risque pas de détruire quoi que ce soit en insérant un mauvais code)

Les fonctions et les réponses attendues par le système sont présentes dans la section mode d'emploi.

II. Description modulaire

Getposition

Fonction : Cette fonction permet de connaître la position lue par l'encodeur *gray*.

Paramètre entrée : néant
Paramètre sortie : octet (*char*) contenant la position décodée.
Variables globales modifiées : néant
Autres
Le paramétrage se fait au début du programme dans les *define* (où est branché cet encodeur)

SendEdata

Fonction : Cette fonction permet d'envoyer 2 octets dans une demi-page de l'EDATA

Paramètre entrée : Adresse de la page (2char), donnée (2char), bit d'offset : détermine si l'on doit écrire dans EDATA1/2 ou EDATA3/4.
Paramètre sortie : bit : 0 : ACK, 1 = NACK
Variables globales modifiées : néant
Autres
Néant

Savelog

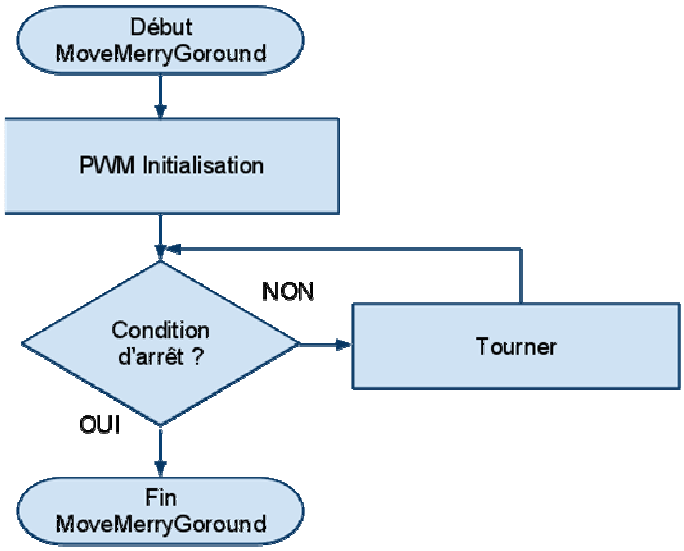
Fonction : Cette fonction permet sauver dans la *FRAM2* les *LOG*

Paramètre entrée : Pointeur des variables de l'horodatage, type de log (couvercle, chauffage,…)
Paramètre sortie : néant
Variables globales modifiées : Offset2 : l'offset total de la *FRAM2*
Autres
Néant

Movemerrygoround

Fonction : Cette fonction permet de tourner d'une position le carrousel.

Paramètre entrée : néant
Paramètre sortie : néant.
Variables globales modifiées : néant
Autres
Utilise les PWM, gère différentes vitesses (paramètre)



Organigramme 2.1 : Fonction MoveMerryGoround, permettant la mise en rotation du carrousel

La condition d'arrêt est : Si on est en face de la barrière infrarouge
Tourner est un peu « simpliste », en fait, pour éviter de l'inertie, on met du couple (des impulsions) pendant un certain temps, puis on fait une pause sans couple. Néanmoins pendant ces pauses, nous vérifions la condition d'arrêt.

OpeningCouvercle

Fonction : Cette fonction permet d'ouvrir le couvercle

Paramètre entrée : néant
Paramètre sortie : *néant*
Variables globales modifiées : néant
Autres
Utilise les PWM, gère différentes vitesses (paramètre). Rem : la fonction homologue : CloseCouvercle existe, et est sur le même principe
Cette fonction est très ressemblante à *movemerrygoround*() ;, sauf qu'il n'y a pas de conditions d'arrêt, on incrémente (ou décrémente) progressivement le rapport cyclique, avec une pause déterminée par l'utilisateur, pour faire tourner en douceur.

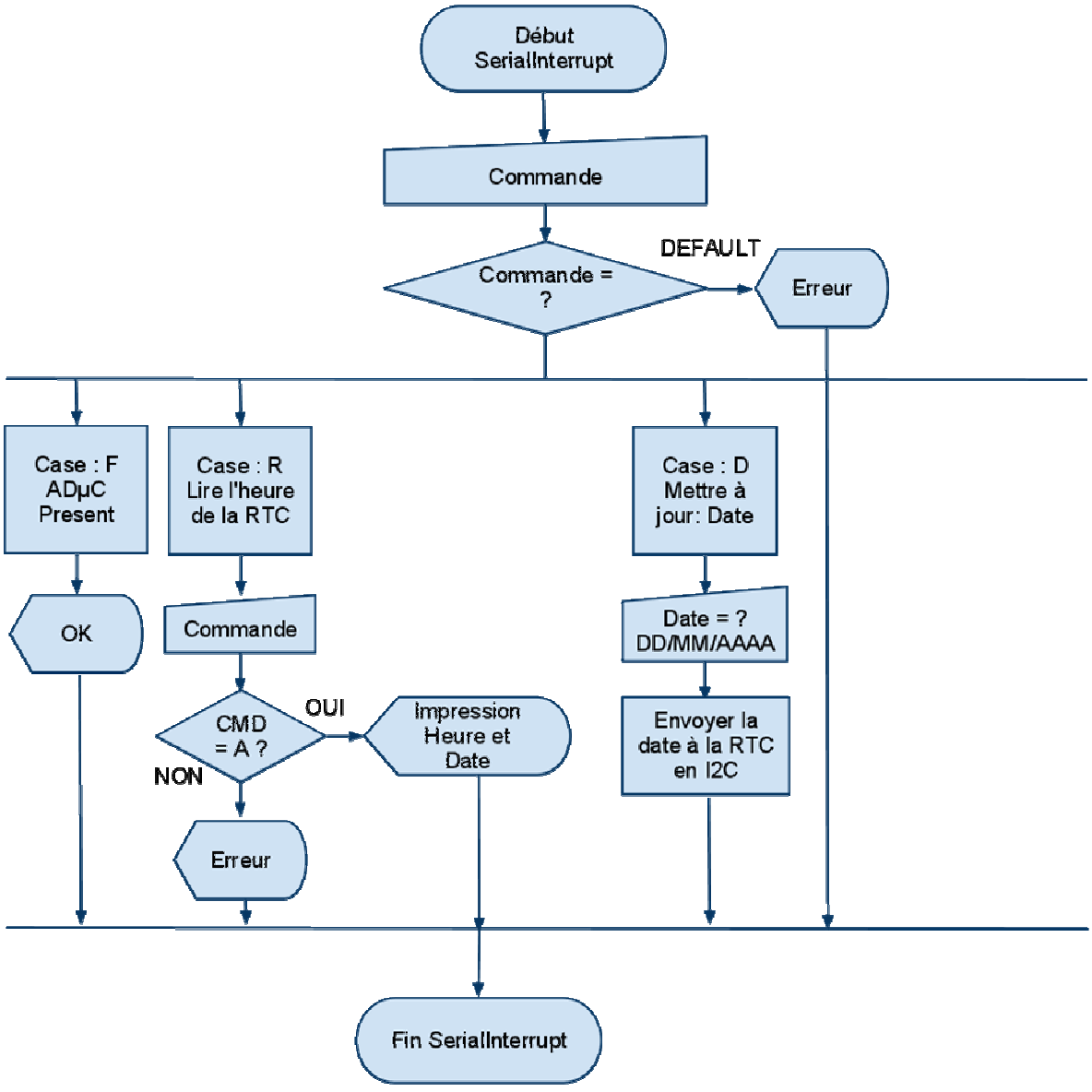
SerialInterrupt

Fonction : Gère l'interruption du port série pour le paramétrage du projet, comme vu précédemment.

Paramètre entrée : néant
Paramètre sortie : *néant*.
Variables globales modifiées : Variable contenant l'horodatage, lors de la lecture de l'heure/date ou de l'envoi de celle-ci. Les variables contenant les divers paramètres énumérés auparavant.

Autres
Néant

Comme on peut le voir sur l'organigramme suivant (qui ne reprend pas tous les paramétrages pour une raison de lisibilité), la commande envoyée par le port série, est recueillie comme *argument* d'un *switch*, une commande correspondant à une action. Lorsque cette commande est effectuée, on ressort du *switch*, et réarmons l'interruption du port série.



Organigramme 2.2 : Une partie de la fonction Serial Interrupt, gérant les interruptions du port série pour le paramétrage du projet.

*TIC*interrupt

Fonction : Cette fonction permet de gérer le service d'interruption de la *TIC* (acquisitions de fréquences tous les 128ème de secondes, time out du pluviomètre, …). Cette fonction décide de l'ouverture ou non du couvercle, et vérifie avant la fermeture due au timeout que le capteur est réarmer (chauffe s'il le faut). Tient compte de l'état de santé de la batterie dans ses actions (ouverture, chauffage, …)

Paramètre entrée : néant
Paramètre sortie : néant.
Variables globales modifiées : fréquence min et max, le *flag* ouverture,
Autres
Modifie l'action du chauffage, peut rester bloquée dedans si la fréquence ne remonte jamais au dessus du seuil de réarmabilité prévu de 12928 Hz.

L'organigramme se trouve à la page suivante.

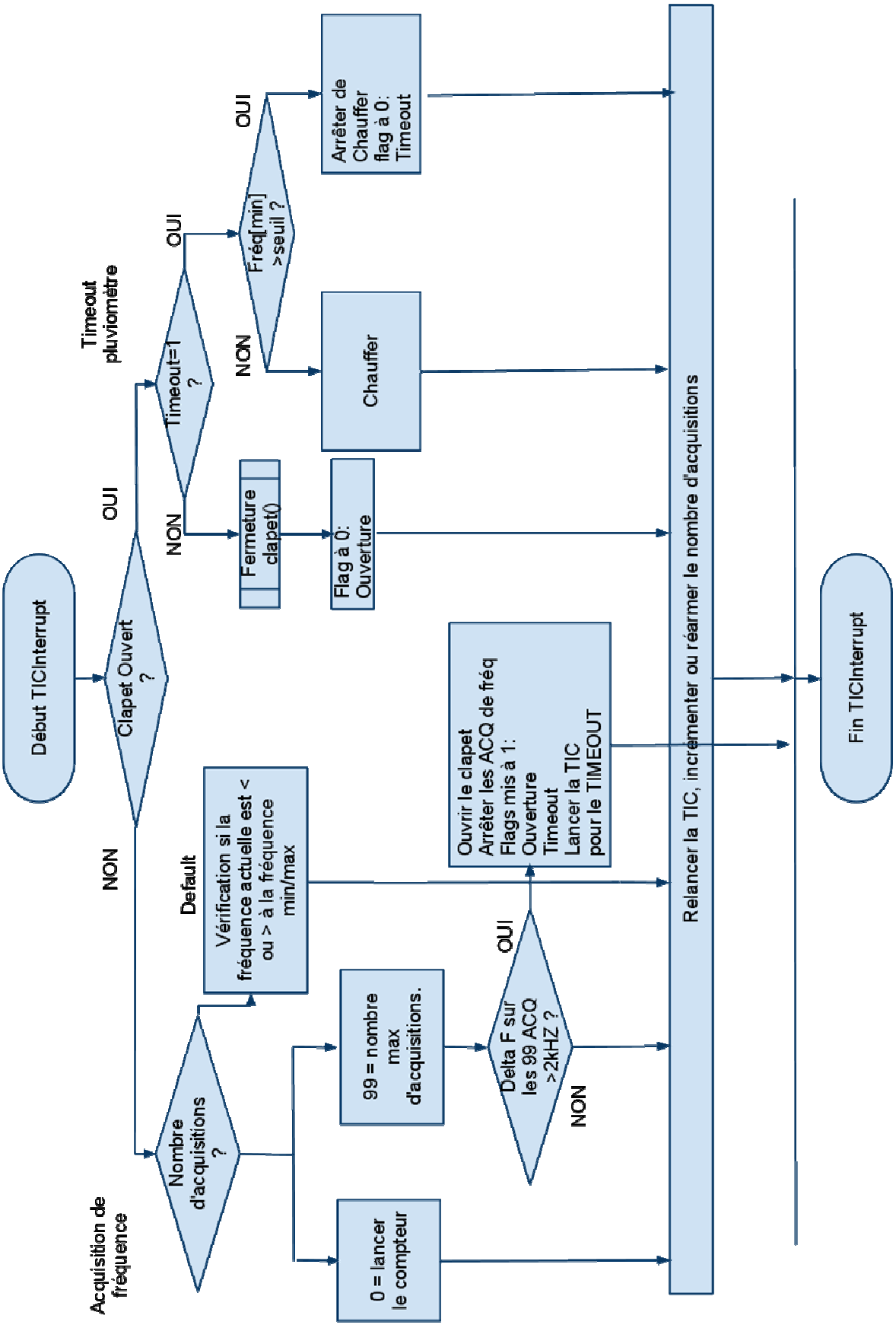
Le programme fonctionne comme suit :
On vérifie si le couvercle est ouvert, si c'est le cas, on est certain d'être en présence d'une interruption pour le timeout sans basculement. Dans le cas contraire, nous sommes toujours en mesure de fréquences.

Dans le cas où l'on mesure la fréquence sur T0 (capteur d'humectation), on fait 100 mesures d'affilée, avec une variable *static* (locale), contenant le nombre d'acquisition de fréquence effectuées. Un *Switch* est présent pour déterminer les actions à effectuer selon le nombre d'acquisitions. Si c'est la première (Nombre d'acquisitions =0), on lance le compteur, tout comme pour les 98 mesures suivantes, mais dans ces 98 mesures, on vérifie si la mesure actuelle est plus grande ou plus petite que la fréquence respectivement maximale et minimale de cette salve de mesures.

A la 100ème mesure, on vérifie si le Δf correspond à la condition de sensibilité (par exemple, 2kHz par défaut), si c'est le cas, et que la batterie est en bonne santé, l'ouverture du couvercle se fait, et le *flag* ouverture est mis à 1.

On fait donc une salve de 100 impulsions permettant d'obtenir un *minimum et un maximum* en termes de fréquence, et d'obtenir un Δf > 2000 Hz (par exemple).

Dans le deuxième cas, où il s'agit du timeout du pluviomètre, on va vérifier avant de fermer si la condition de ré-armabilité est obtenue (fréquence assez haute). Dans le cas contraire, on va chauffer jusqu'à obtenir satisfaction, on relance la *TIC* pour vérifier en boucle la fréquence.



Organigramme 2.3 : Fonction TICInterrupt permettant de cadencer les acquisitions de fréquence et de gérer le timeout

Convad10

Fonction : Cette fonction permet de convertir en digital sous 10bits

Paramètre entrée : canal à convertir
Paramètre sortie : valeur brute de conversion, sous 10bits
Variables globales modifiées : néant
Autre : néant

Calcultemp

Fonction : Cette fonction permet de calculer la température via la CTN sur le capteur de pluie. Est appelée après une conversion analogique-digitale

Paramètre entrée : Résultat de la conversion sur 10 bits
Paramètre sortie : la température en *float*, en °C.
Variables globales modifiées : néant
Autres : Néant

Initialisation

Fonction : Cette fonction permet d'initialiser le processeur, et tous les périphériques liés :

- Ports inutilisés mis en entrée
- Ports utilisés mis dans le sens adéquat
- Initialisation des paramètres
- Initialisation de l'UART
- Initialisation de l'DATA, et écriture d'une valeur particulière
- Initialisation de l'I2C et de ses périphériques :
    - RTC
    - IO Expander
    - RAM
- Initialisation des convertisseurs analogiques-digitaux
- Initialisation des flags globaux

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : au premier reset : tous les paramètres sont mis en valeur par défaut (sensibilité, pause, timeout pluviomètre, quantification, …), les offsets sont également positionnés sur 0. Dans le cas contraire (deuxième reset), le programme va charger ces valeurs enregistrés à l'initialisation dans la EDATA.
Flags globaux initialisés.

Autres
Néant

Scrutint

Fonction : Cette fonction permet de connaître quelle est la source qui a demandé l'interruption.

Paramètre entrée : néant
Paramètre sortie : Néant
Variables globales modifiées : variables de l'horodatage. Lorsque la source a été trouvée, une date/heure lui est associée. Le *flag* correspondant à l'interruption source est également modifié, et le nombre de tubes « vierges » peut-être modifié lorsque le bouton adéquat est appuyé.
Autres : Néant

L'organigramme est à la page suivante.

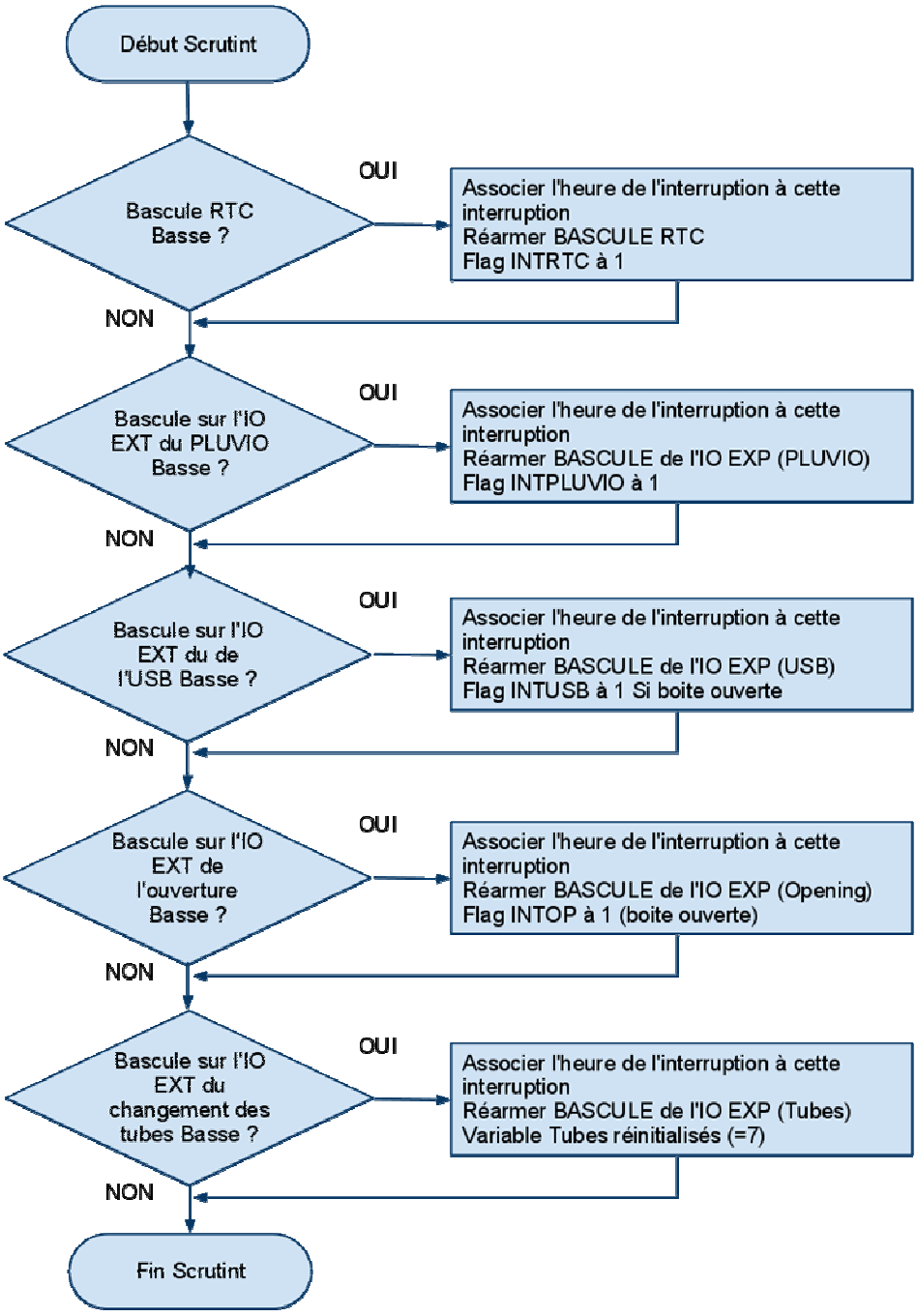
Le programme va d'abord vérifier si ce n'est pas la *RTC* qui a demandé l'interruption (synchrone), si tel est le cas, l'heure enregistrée dans des variables globales lors de l'exécution de l'interruption 0 sera affectée dans un tableau de variables prévues pour l'horodatage de la *RTC*. Le fait de réserver un tableau de variables d'horodatage par interruptions permet d'avoir plusieurs sources d'interruptions « en même temps », ayant le même horodatage. Un *flag* est dès lors mis à 1.

Ensuite, on vérifie si ce n'est pas le pluviomètre qui a demandé l'interruption, via l'état des bascules de *l'IO Expander*. Si tel est le cas, un *flag* est mis à 1, et, comme pour la *RTC*, une série de variables destinées à horodater les interruptions du pluviomètre sont affectées de l'heure et la date à l'origine de l'interruption 0. La bascule en question est réarmée.

Et ainsi de suite pour les trois autres sources.

Seule la source d'interruption « changement de tubes » ne modifie pas de *flag*, et ne demande pas d'exécution d'un autre programme par le *main* (voir plus bas). Cette source initialise directement une variable globale « tubes » (à 7, puisque 7 utilisable, le huitième servant de témoin)

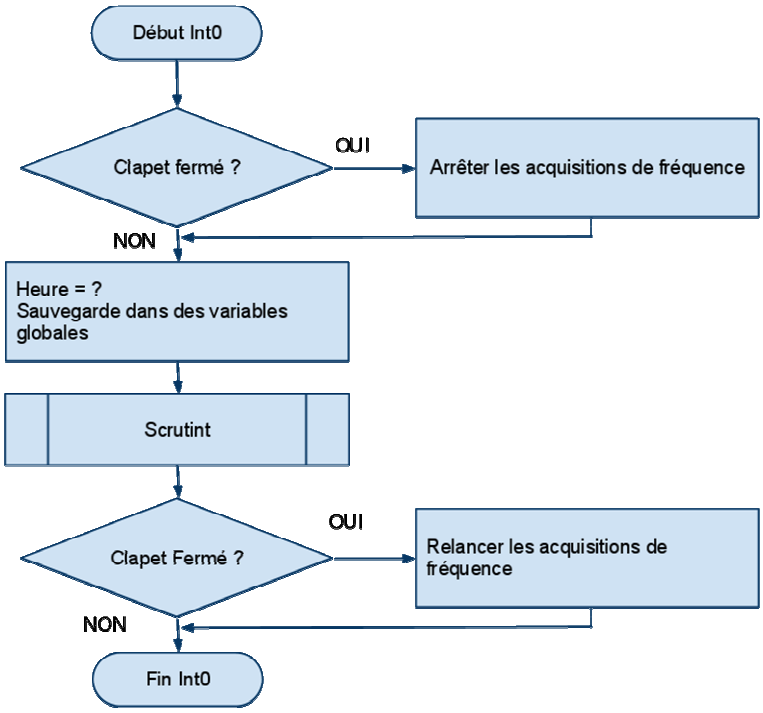
L'ordre de scrutation n'a aucune importance pour déterminer la priorité, car on sauve l'heure au début de la demande d'interruption.


Début Scrutint
Bascule RTC Basse ?
OUI
Associer l'heure de l'interruption à cette interruption
Réarmer BASCULE RTC
Flag INTRTC à 1
NON
Bascule sur l'IO EXT du PLUVIO Basse ?
OUI
Associer l'heure de l'interruption à cette interruption
Réarmer BASCULE de l'IO EXP (PLUVIO)
Flag INTPLUVIO à 1
NON
Bascule sur l'IO EXT du de l'USB Basse ?
OUI
Associer l'heure de l'interruption à cette interruption
Réarmer BASCULE de l'IO EXP (USB)
Flag INTUSB à 1 Si boite ouverte
NON
Bascule sur l'IO EXT de l'ouverture Basse ?
OUI
Associer l'heure de l'interruption à cette interruption
Réarmer BASCULE de l'IO EXP (Opening)
Flag INTOP à 1 (boite ouverte)
NON
Bascule sur l'IO EXT du changement des tubes Basse ?
OUI
Associer l'heure de l'interruption à cette interruption
Réarmer BASCULE de l'IO EXP (Tubes)
Variable Tubes réinitialisés (=7)
NON
Fin Scrutint

Organigramme 2.4 : Fonction scrutInt permettant de vérifier quelle est la source qui demande l'interruption

Inter0

Fonction : Cette fonction est appelée lors d'une demande d'interruption. Cette dernière va chercher l'heure et la date de son exécution, puis va appeler *scrutint* afin de connaitre la ou les source(s) à l'origine de l'interruption.
Paramètre entrée : néant
Paramètre sortie : Néant
Variables globales modifiées : Variables pour l'horodatage
Autres : Néant



Organigramme 2.5 : Fonction Inter0 qui va vérifier, via ScrutInt quelle est la fonction qui a demandé l'interruption

Lors de l'exécution de l'interruption 0, il est important de savoir si le couvercle est ouvert ou fermé. S'il est ouvert, la *TIC* servant pour le *timeout* et il ne faut pas la couper. S'il est fermé, la *TIC* sert au cadencement du comptage de la fréquence sur *T0*, il faut donc le couper, car la *TIC* cadençant la lecture de T0 à 1/128ème de secondes, l'exécution de *scrutint*(); (avec de l'I2C) nécessite de bloquer les sources d'interruptions. Le processeur ne sera pas informé à temps voulu d'aller vérifier la valeur du compteur, et, après l'exécution de *scrutint*(); (plusieurs centaines de millisecondes), la valeur de T0 sera complètement fausse, et l'ouverture du couvercle se fera, à tort.

Après avoir scruté quelle est la source qui a demandé l'interruption, on doit relancer la *TIC* seulement si le couvercle n'est pas ouvert, sinon ne pas toucher à la *TIC* (qui détermine le timeout) !

RTCI

Fonction : Service d'interruption de la *RTC*. Lorsque celle-ci demande, *Inter0* va appeler *Scrutint*, et mettre le flag INTERRTC à 1. Le *main* lancera cette fonction. Cette fonction s'occupe de faire les acquisitions synchrones (toutes les dix secondes, cadencée par la *RTC*).

Paramètre entrée : néant
Paramètre sortie : Néant
Variables globales modifiées : le *flag ouverture*, lorsqu'il est minuit et que le filtre est changé, offset (variable pointeur de la zone mémoire en *FRAM1* utilisée), bit de l'interruption de la *RTC*
Autres : Néant

L'organigramme se trouve à la page suivante.

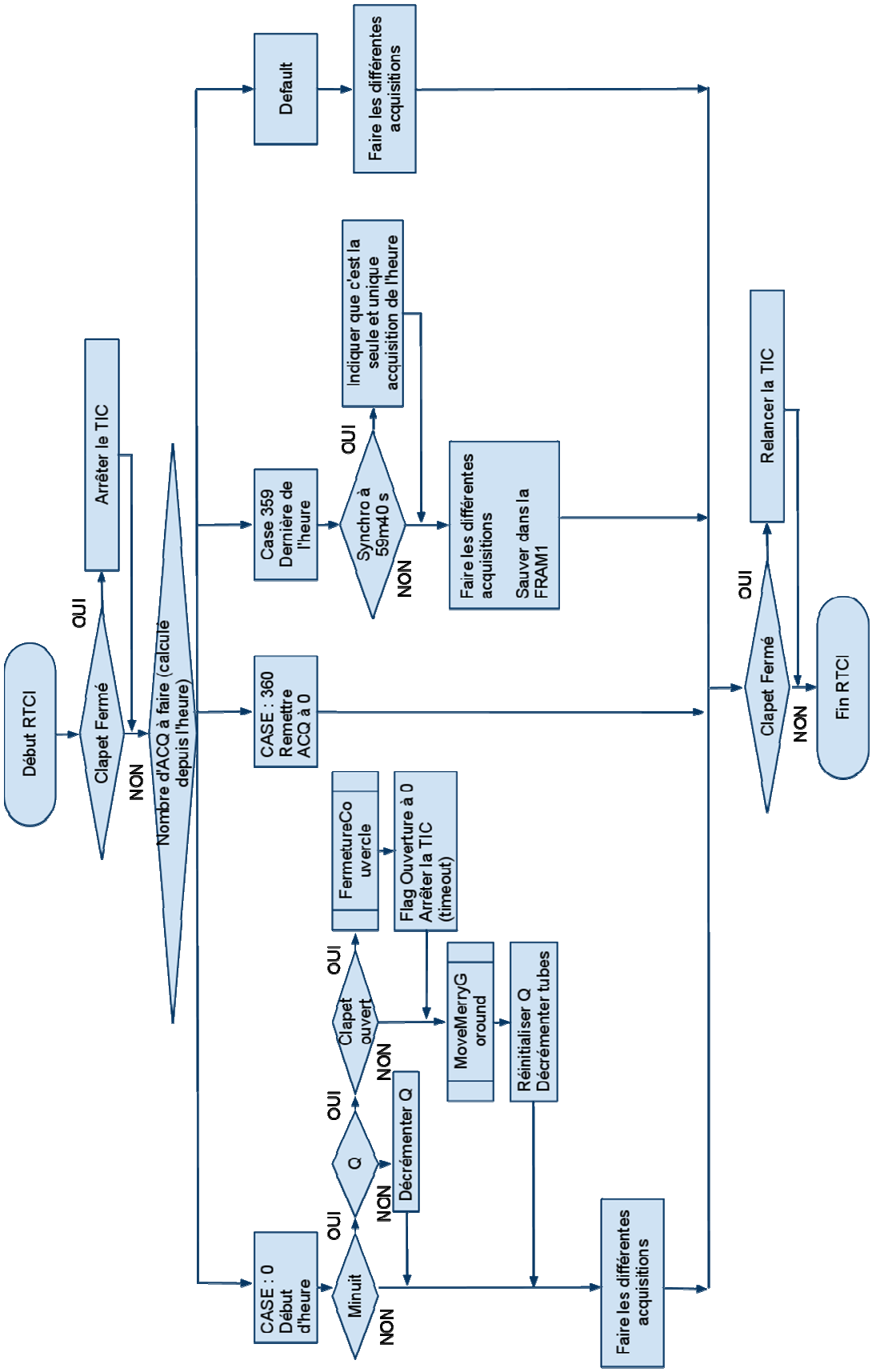
Dans un premier temps, nous vérifions si le couvercle est fermé, si c'est le cas, c'est que nous sommes en train de faire des mesures de fréquence, cadencée par la *TIC*, il faut donc arrêter la *TIC* et le compteur T0, afin d'éviter lorsque la fonction *RTC*I soit finie que la valeur du compteur soit erronée et fasse ouvrir à tort le couvercle.

Nous allons vérifier, selon l'heure qu'il est, combien d'acquisitions auraient du être faites, sachant qu'il y a 3600 secondes dans une heure et que l'on fait des acquisitions toutes les secondes, il y aura donc 360 acquisitions par heure.

Dans la phrase ci-dessus, je note « combien d'acquisitions auraient du être faites », c'est dans le cas où l'on a commencé les acquisitions à une heure pile (par exemple 12h00m00s). Par exemple, si on installe le projet à 12h50m00s, il reste 6 acquisitions à faire avant 13h, mais 0 ont été faites (première mise sous tension).

Si on est en début d'heure (0 acquisitions faites, et 0 qui auraient du être faites), on vérifie que l'heure vaut zéro. Dans ce cas, c'est qu'il est minuit. Si la condition de quantification est respectée, on tourne le carrousel, plus concrètement, si l'on demande 1 tube tous les deux jours, le 1^{er} jour, le programme ne va pas tourner le carrousel, mais décrémenter une variable « Q », le deuxième jour, cette variable contiendra la condition pour tourner, et nous tournerons donc. Avant de tourner, on vérifie si le couvercle est ouvert (il pleut donc…), on referme le couvercle, et on arrête la *TIC* qui sert pour le timeout (puisque on a refermé). Si après cette exécution, il pleut toujours, le couvercle se rouvrira, avec le nouveau filtre / tube.

Lors de la dernière acquisition de l'heure (case 359), on va sauver dans la *FRAM1* et calculer les moyennes. Néanmoins, si on a remis à l'heure à 12h59m40s, nos calculs montreront qu'on devrait être à 359 acquisitions, mais nous n'en avons en réalité aucune, on testera donc une variable locale (*static*), contenant le nombre d'acquisitions réellement réalisées dans l'heure, si cette variable contient 0, on sait déterminer qu'on a remis à l'heure à 59m40s.
Le *case Default* s'occupe des 357 autres cas d'acquisitions, entre 00m10s et 59m40s.
Lorsque l'on sauvegarde dans la FRAM, on initialise toutes les variables de débits et cumuls, car ceux-ci portent sur l'heure.



Organigramme 2.6 : Fonction RTCI qui cadence les acquisitions synchrones, et sauve quand il le faut les données d'une heure dans la FRAM.

Néanmoins, à l'écriture de ce TFE, une nouvelle solution a été trouvée. Celle-ci n'a toutefois pas été testée.

L'organigramme se trouve à la page suivante.

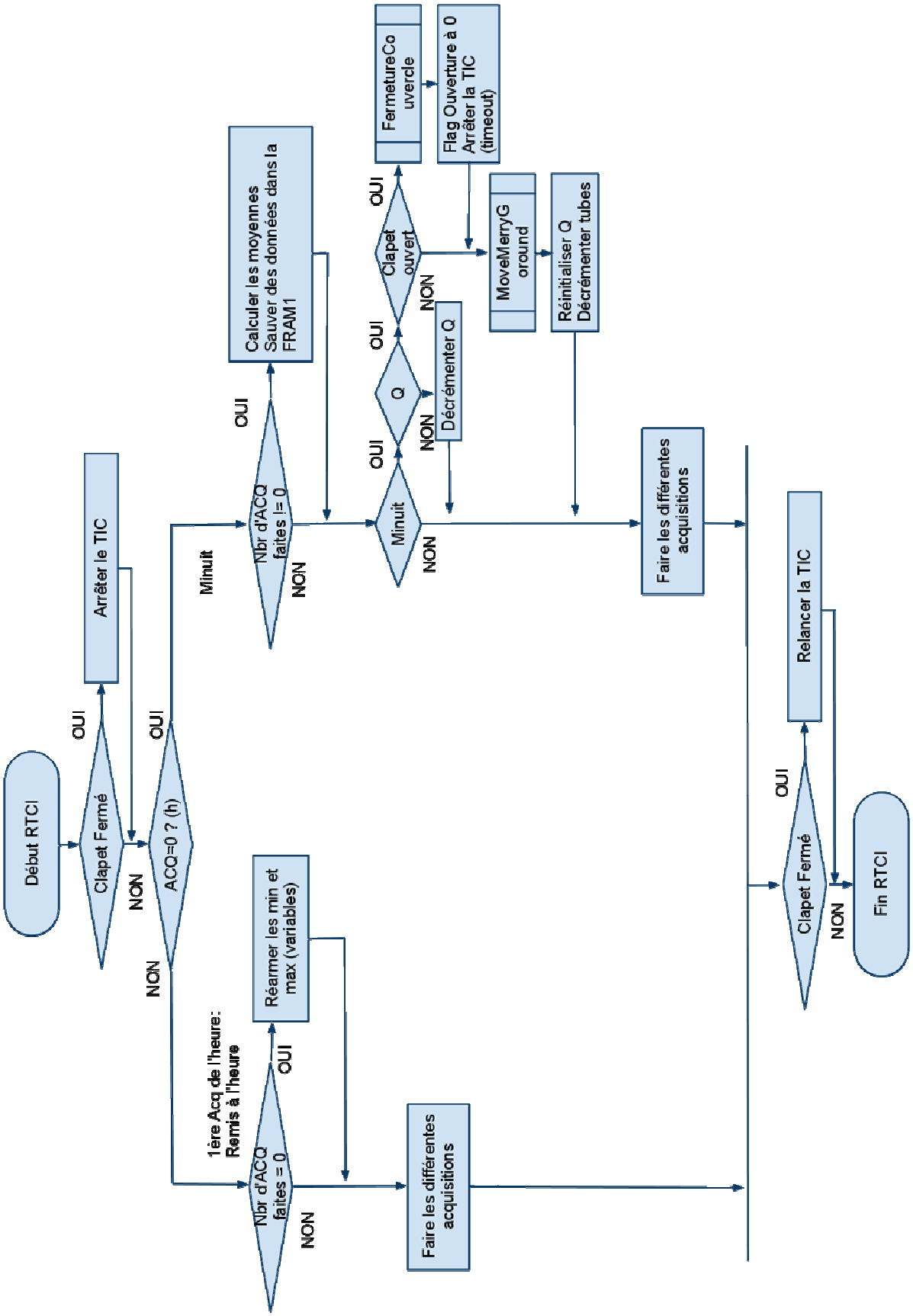
On va toujours tester le nombre d'acquisitions qui auraient dû être faites en fonction de l'heure (si il est midi, aucune auraient du être faites). Si ce nombre d'acquisition est bien nul, et que le nombre d'acquisitions effectives vaut bien zéro, nous sommes en début d'heure et nous allons sauver sur la *NOVRAM1* les données de l'heure. Si le nombre d'acquisitions étaient différent de 0, cela signifierait qu'on aurait remis à l'heure. La différence entre ici et avant, est qu'on fait cette étape en début d'heure.

Avant, on le faisait à la dernière acquisition, et on sauvait par exemple à 12 :59 :50 secondes. Ce qui se passait entre cette heure là, et 13h00, était perdu.

Le reste demeure sensiblement identique

L'avantage principal est le gain d'espace code, et une moyenne sur l'heure complète ! (361 valeurs, contre 360)

REM : Ceci n'a pas été testé !



Organigramme 2.6b : Fonction RTCI qui cadence les acquisitions synchrones, et sauve quand il le faut les données d'une heure dans la FRAM – non testé

*USB*I

Fonction : Cette fonction se charge d'écrire sur la clé *USB* les données, ainsi que les *LOGs*, présents sur les deux *FRAM*. Cette fonction permet aussi de déterminer si l'utilisateur veut charger les logs sur sa clé *USB*, ou brancher son PC.

Paramètre entrée : néant
Paramètre sortie : Bit permettant de dire si il y avait une clé *USB* présente, permettra de définir si l'utilisateur désirait charger sur sa clé *USB*, ou brancher son PC
Variables globales modifiées : Offset, offset2, buffer du *scanf,* bit de l'interruption de l'*USB*
Autres
Utilisation de la fonction *SPRINTF* permettant de quantifier le nombre d'octets à envoyer à la clé *USB*.

L'organigramme est à la page suivante.

Dans un premier temps, il nous faut vérifier si la clé *USB* est présente, si tel n'est pas le cas, la demande d'interruption (*USB*) est alors faite pour brancher un PC. Il faut retourner un flag (ret) donnant cette information au programme appelant.

Si la clé est présente, il faut vérifier si le fichier donnée « CRADATA.xls » existe, dans la négative, le créer, avec son header :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Date	Temp min [°C	Temp moy [°	Temp max [°	Dew min [°C]	Dew moy [°C	Dew max [°C	HR min [%]	HR moy [%]	HR max [%]	Bat moy [mV	Debit min [m	Debit moy [m	Debit max [m	Cumul [mm]
2															
3															

Ensuite, on va charger les données sur la clé *USB*, pour ce faire, on va aller chercher dans la *FRAM* toutes les valeurs d'une ligne (=une heure, en général) ce qui constitue :

- 6 octets pour l'horodatage
- 2 octets par températures (minimale, moyenne, maximale)
- 2 octets par point de rosée (minimal, moyen, maximal)
- 1 octet par taux d'humidité relative (minimal, moyen, maximal)
- 2 octets pour la tension de la batterie (mV)
- 2 octets par débits (minimal, moyen, maximal)
- 2 octets pour le cumul

Soit un total de 31 octets par lignes.

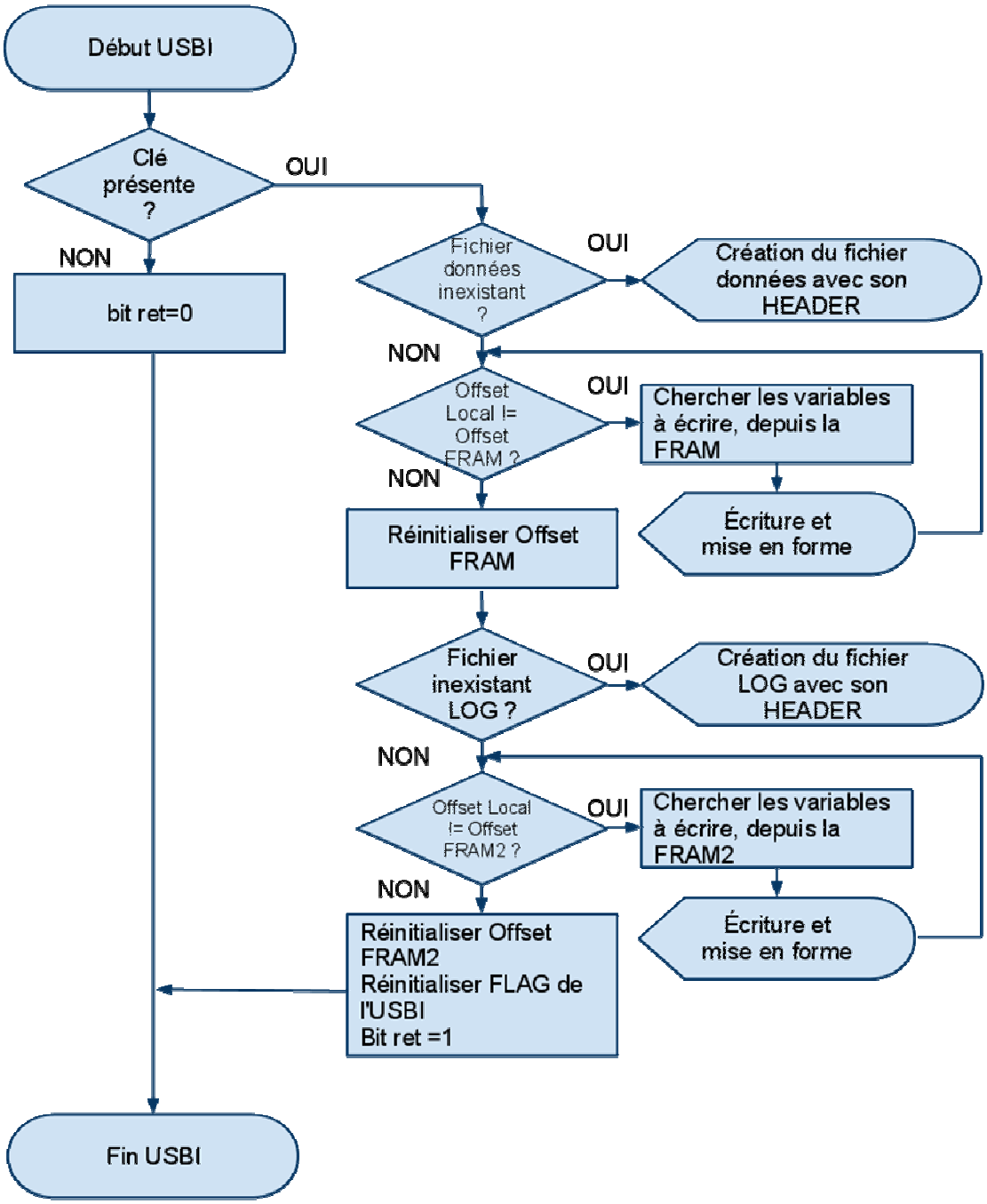
On va à chaque lignes incrémenter notre offset local de 31, et lorsque l'offset local sera = à l'offset de la FRAM1 (ou 2, pour le fichier LOG), ceux-ci seront remis à zéro, et toutes les données seront dès lors envoyés :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Date	Temp min [°C	Temp moy [°	Temp max [°	Dew min [°C]	Dew moy [°C	Dew max [°C	HR min [%]	HR moy [%]	HR max [%]	Bat moy [mV	Debit min [m	Debit moy [m	Debit max [m	Cumul [mm]
2	30/04/2010 -	22.1	22.1	22.2	8.0	8.0	8.1	40	40	40	11688	0.0	0.0	0.0	0.0
3	30/04/2010 -	22.1	22.1	22.1	7.9	7.9	7.9	40	40	40	11596	0.0	0.0	0.0	0.0

Note : Pour contrer une erreur lors de l'enregistrement de l'offset d'une des FRAM et/ou des données, nous vérifions que cet offset soit bien une valeur multiple de 31, dans le cas contraire, on le tronque, afin de ne pas compromettre le bon déroulement du programme (*while(offsetlocal !=offsetRAM)*)

Le principe est totalement identique pour le fichier LOG, à la différence prête où la donnée dans le cas du fichier log se constitue de l'horodatage ainsi que d'un argument (un chiffre) désignant le type d'évènement (7octets / log).

Nous vérifions uniquement cette variable, et selon son type, nous envoyons la bonne mise en forme (via un *switch*), non représenté sur cet organigramme pour cause de lisibilité.



Organigramme 2.7 : Fonction USBI qui permet le chargement sur clé USB des données

Pluvioi

Fonction : Lorsque *Scrutint* a déterminé que c'était le pluviomètre qui demandait l'interruption (basculement), *PLUVIOI*() ; est appelée via le *main (flag*), cette fonction calcule les débits et le cumul. En fin d'heure, la fonction *RTCI* sauvegardera ces données dans les *FRAM*.

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : variables de l'horodatage, du cumul et des débits, flag de l'interruption du pluviomètre
Autres
Néant

L'organigramme se trouve à la page suivante

S'il s'agit du premier basculement de *l'heure*, on va tout simplement faire la partie commune, qui est de sauver l'horodatage, incrémenter le cumul, réarmer le relancer la *TIC* (timeout). On incrémente une variable qui contient le nombre de basculements détectés dans l'heure (remise à 0 dans *RTC*I à la fin de l'heure)

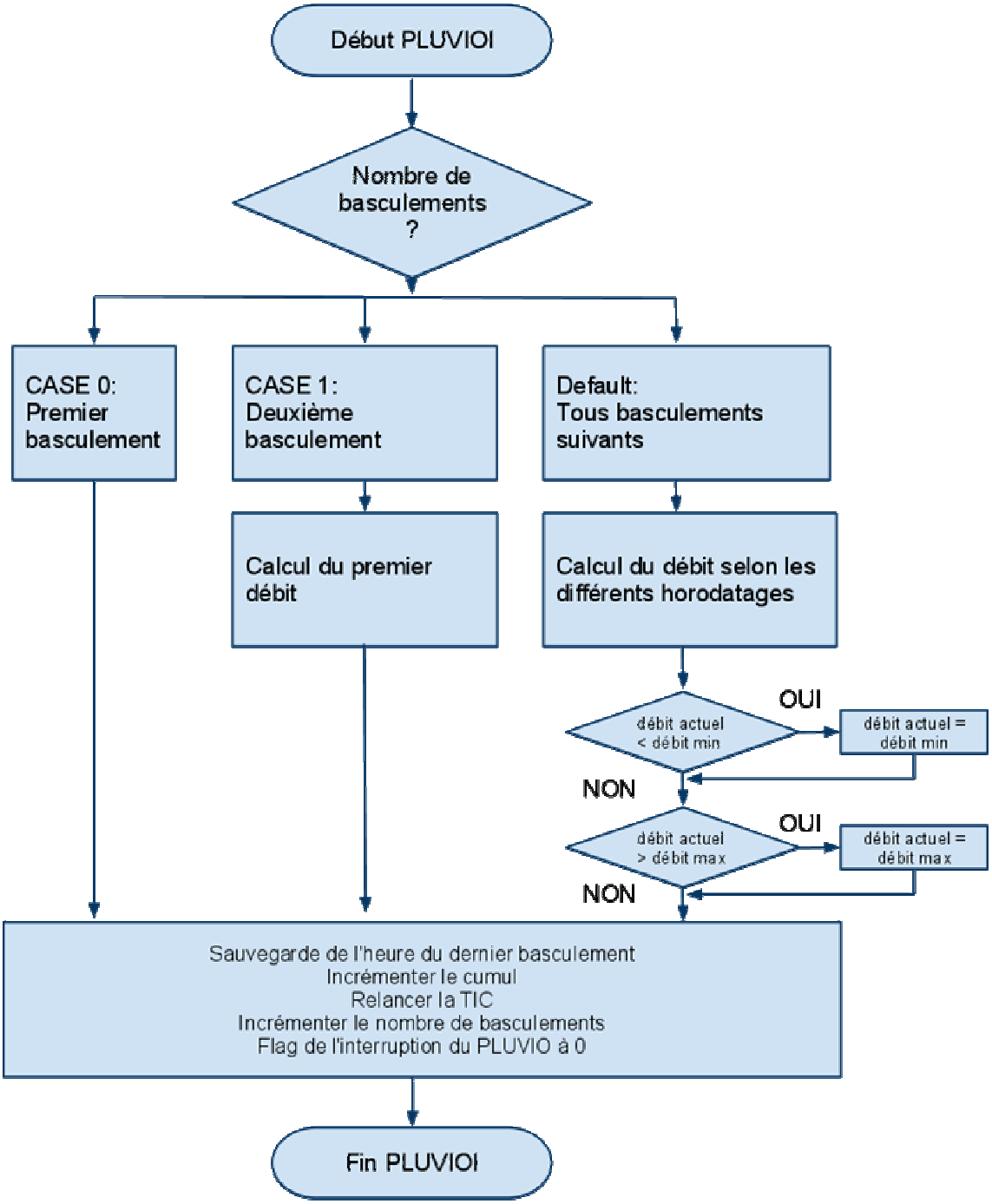
Le basculement suivant (le deuxième), permet de calculer le premier débit selon la formule suivante :

$$débit = \frac{auget}{\Delta t} 3600 \ [mm/h]$$

Où Auget est défini par le nombre de mm pour lequel l'auget bascule.

Les basculements suivants (cas général), le programme calculera le débit courant, et on le compare au débit minimal et maximal de l'heure.

Fournir les données météorologiques relative à la pluie en temps réel n'est toutefois pas possible, pour des raisons indépendantes de notre volonté, à savoir que les interruptions sont bloquées pendant plusieurs centaines de millisecondes, lorsque *l'I2C* est utilisé, de ce fait les centièmes de secondes ne sont plus justes, si le basculement vient lorsque *l'I2C* est en utilisation, néanmoins notre programme donne une idée très précise de cette pluie.



Organigramme 2.8 : Fonction PLUVIOI qui permet de calculer la pluviométrie (débit, cumul)

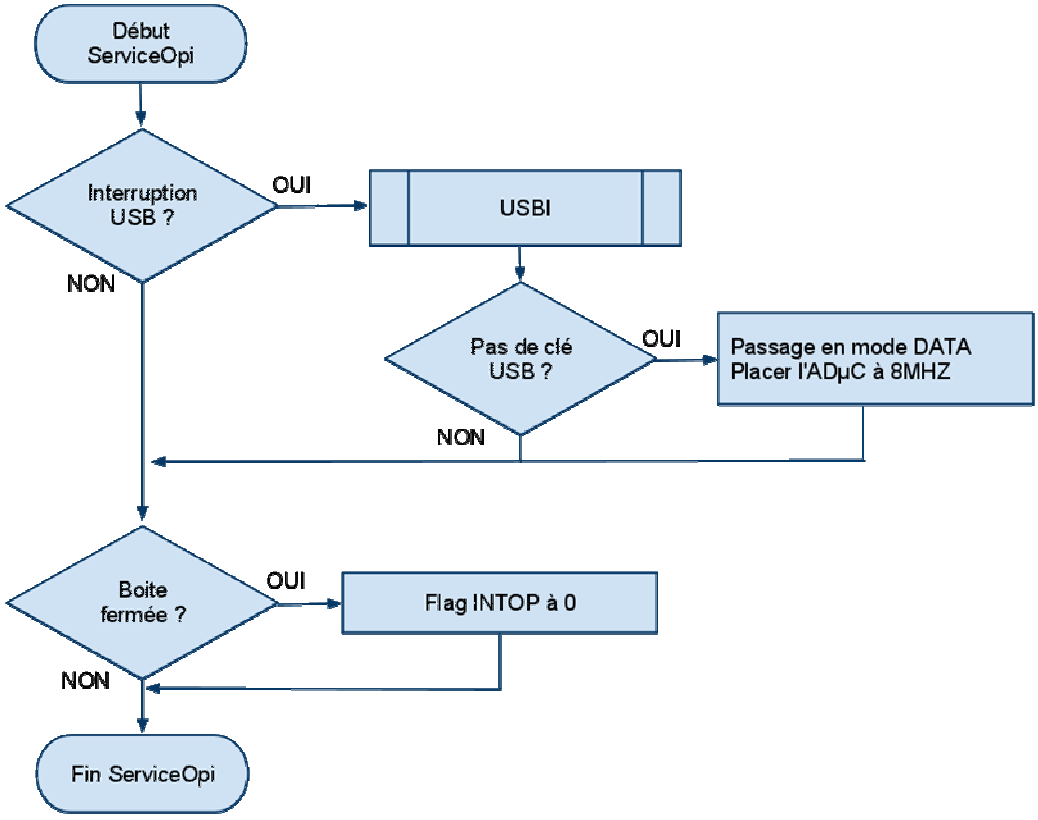
Serviceintopi

Fonction : Fonction qui va se charger de déterminer, lors de l'ouverture du projet, si on veut charger les données sur la clé, ou si on désire brancher un PC sur le projet, gère les deux modes (passage de l'un à l'autre). Est appelée par la fonction *OPI*, lorsqu'une demande a été détectée par *scrutint* (bouton *usb*)

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : bit de l'interruption de l'ouverture, lors de la fermeture de la boîte.
Autres
Néant

Cette fonction va vérifier en premier lieu si l'interruption n'est pas demandée par l'*USB*. Si tel est le cas, elle va appeler *USBI()*; , qui va se charger de vérifier si la clé est présente, un *bit* sera retourné. Si la clé est présente, le chargement se fera. Si pas, il faudra passer en mode UART/DATA, et passer à 8MHz temporairement, pour l'interface en C#.

Si cette fonction a été appelée car la boîte s'est fermée, on met le *flag* global « *INTOP* » à 0, libérant le *while* principal de la fonction *OPI*() ;



Organigramme 2.9 : Fonction ServiceIntOPI qui permet de lancer le chargement sur la clé USB, ou l'initialisation en mode DATA (PC)

*VDIP*ION

Fonction : Fonction qui va se charger d'initialiser le *VDIP2*, en mode *ASCII* et en mode *Short* (voir plus bas). Allume la *LED* d'activité de l'*USB* en vert

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : Buffer général du scanf.
Autres
Néant

OPI

Fonction : Fonction qui va se charger de gérer l'ouverture de la boîte : *LED Low battery*, appel de *VDIPON();*, arrêter l'alarme (10sec) de la *RTC*, ferme le couvercle en cas de pluie détectée auparavant, lance *serviceintopi()*; lorsqu'une demande via le bouton *USB* du panel est détectée (*scrutint()*;). Lorsqu'on referme le capteur, un log est sauvé (tout comme à l'ouverture), et on repositionne sur une valeur cohérente (barrière infrarouge) le carrousel. On resynchronise la *RTC* sur un multiple de 10secondes, et relance l'alarme toutes les 10secs.

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : Buffer général du scanf, flag ouverture (couvercle) quand il le referme, offset2 via *savelog()*;,
Autres Néant

L'organigramme se trouve à la page suivante

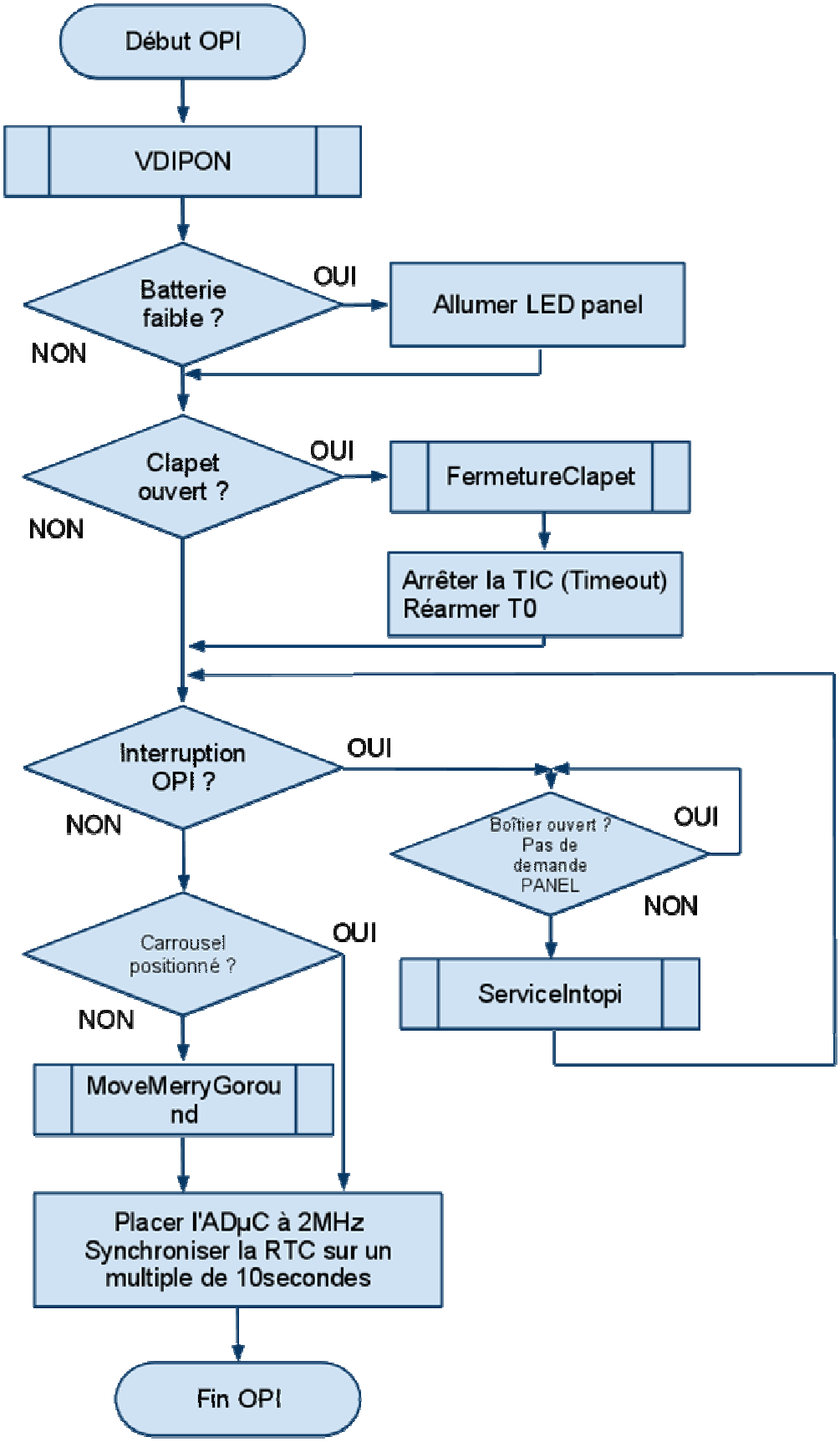
Dans un premier temps, on initialise le *VDIP2*, puisque ce service d'interruption se lance en même temps que la mise sous tension de ce module. On vérifie la tension de la batterie, si celle-ci est faible, on allume la *LED* du panel pour prévenir le technicien venant changer les filtres.

Si le couvercle est ouvert, il faut le fermer, et arrêter le timeout.

Ensuite, le programme se compose principalement de deux *while*, le premier :
While(INTOPI), tant que l'interruption de *l'opening* est active, faire le *while* suivant :
While(Ouvert ou Pas d'interaction du panel) ne rien faire. Lorsque l'utilisateur appuie sur le bouton *USB* du panel, la fonction « *ServiceIntopi* » va vérifier si l'on désire charger les données sur une clé *USB*, brancher le PC ou même si on a refermé la boîte.

En ayant refermé la boîte, on repositionne sur une valeur cohérente le carrousel, et on synchronise la *RTC* sur un multiple de 10secondes.

On place également à 2MHz la *PLL* de *l'ADμC*, car celle-ci peut avoir été modifiée pour la demande de passage en mode *DATA*. Si elle n'a pas été modifiée, aucune influence.

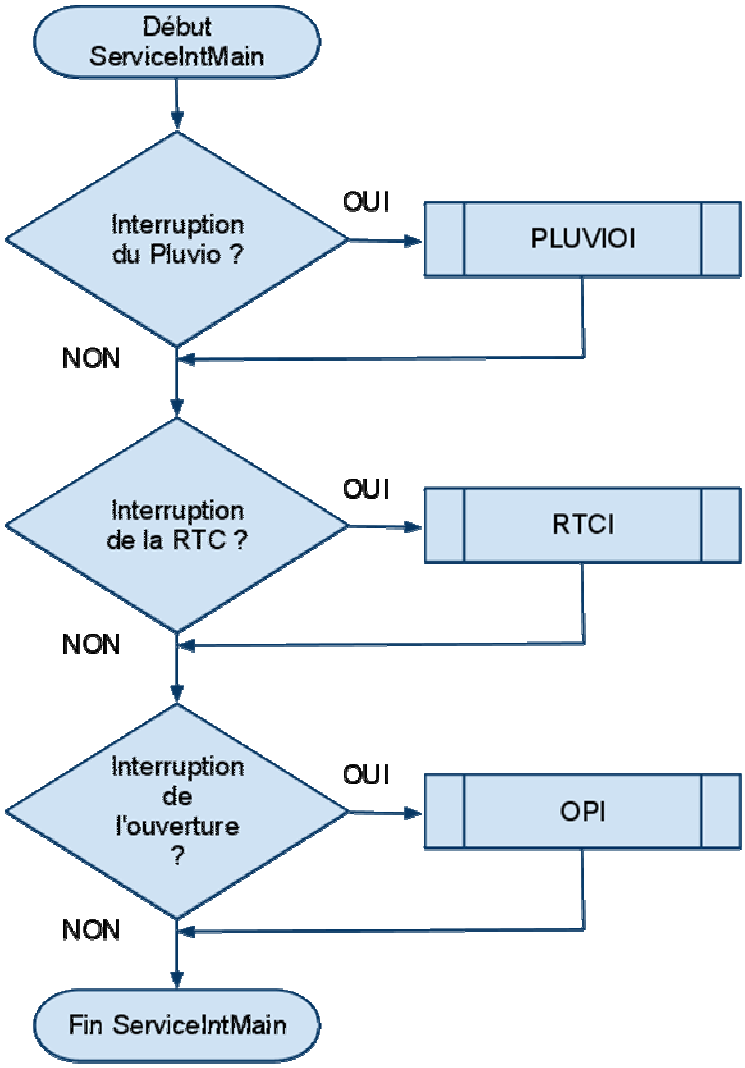


Organigramme 2.10 : Fonction OPI qui permet de gérer l'interruption lors de la maintenance

Servicemainint

Fonction : Fonction qui tourne en boucle dans le main, permettant de lancer les fonctions *PLUVIOI*();, *RTCI*(); et *OPI*();, lorsque les *flags* respectifs sont mis à 1 par *scrutint()*;

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : néant
Autres
Néant



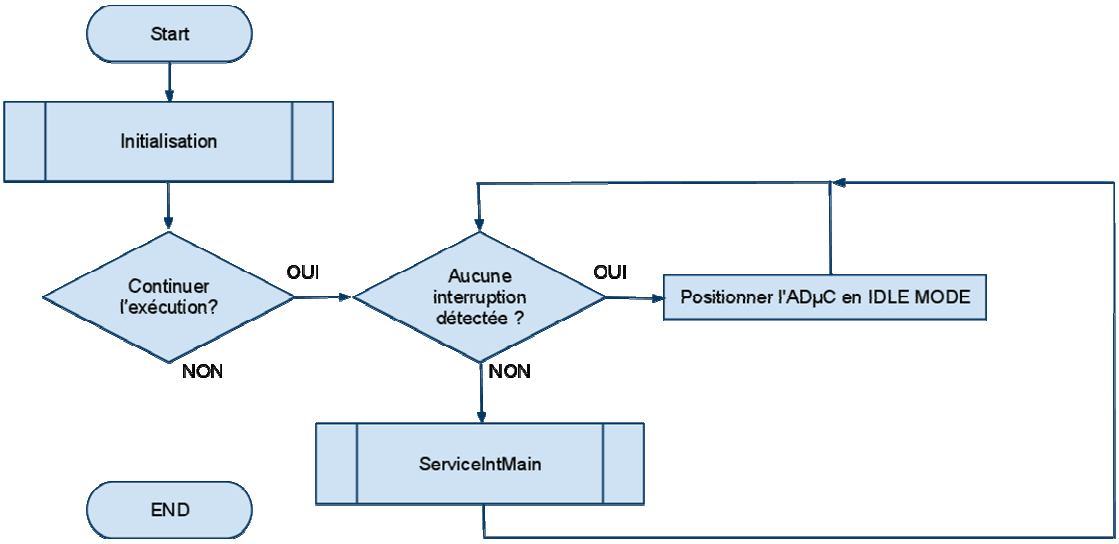
Organigramme 2.11 : Fonction ServiceIntMain qui permet de lancer les bons services d'interruptions

Cette fonction va simplement vérifier les différents *flags* globaux, si ceux-ci ont été mis à « 1 » par *scrutint();*, cette fonction va lancer la fonction correspondante.

Main

Fonction : Fonction principale, lançant *l'initialisation*();, et *servicemaintin*() ;

Paramètre entrée : néant
Paramètre sortie : néant
Variables globales modifiées : néant
Autres
Néant



Organigramme 2.12 : Fonction Main

Il s'agit donc de deux boucles « *tant que » (while*), faire l'action tant que la condition est vraie. Le premier, est caractéristique à la programmation embarquée, qui est un *while*(1) qui sera toujours vrai, le deuxième, vérifie qu'un des trois flags pouvant lancer un des trois services d'interruptions est vrai. Tant qu'aucune interruption n'est détectée par *l'ADµC*, on le laisse en *Idle Mode*, pour la consommation. Lorsqu'une des interruptions est détectée, *Scrutint()*; va positionner le flag adéquat à 1, rendant la condition favorable au lancement de *ServiceIntMain();*, qui va charger le bon programme correspondant à l'interruption demandée.

III. Programmation du module spécifique : VDIP2

Le Module *VDIP2* de *FTDI*, vu précédemment, se commande via le port série, et les fonctions dans la librairie <*stdio.h*> *printf(); et scanf();*

Il faut néanmoins noter en premier lieu un problème inhérent à la fonction *scanf* qui est l'écho. Cette fonction, via d'autres appels de fonctions, est amenée à un moment à réécrire sur le port série le caractère qu'il vient de scanner (via *putchar*).

Nous avons réécrit *putchar*, selon les sources sur keil.com, afin de ne plus avoir ce problème.

Le *datasheet* fourni par *FTDI* était de mauvaise qualité point de vue détails des commandes. En fait, lorsque l'on envoie une commande, une réponse est attendue. Dans le mode étendu, cette réponse a son nombre de caractères variant. Nous avons dans un premier temps cherché sur internet des exemples de programmes. Tous les programmes trouvés étaient pour un *PIC* et en *Basic*. Les auteurs ne faisaient pas de *scanf*, mais simplement « tant qu'il y a quelque chose qui rentre, ne rien faire ». Peu pratique si l'on veut faire des tests, et s'assurer de l'intégrité de la transmission.

En premier lieu, lorsque le *VDIP2* s'initialise, celui-ci envoi au microcontrôleur une chaîne de caractères dont la taille est 25 ou 60. Ceci dépend de si une clé *USB* est connectée ou non lors de l'envoi. Cette chaîne se compose comme ceci :

```
<CR>
Ver·03.68VDPSF·On-Line:<CR>
```

Soit 25 caractères (*char*). Lorsqu'une clé *USB* est introduite, avant le démarrage, des informations sur la présence de celle-ci, ainsi que la possibilité d'upgrade du *firmware* est transmis, portant la longueur du *string* à 60 caractères.

Le *VNC1L* répond à deux modes de commandes comme expliqué dans la partie interfaçage :

- Mode court (*Short Command Set*)
- Mode étendu (*Extended Command Set*)

Nous avons dans un premier temps testé par approches successives chaque commande du mode étendu, en voyant combien de caractères renvoyait le *VDIP2*. Nous n'avons pas testé toutes les fonctions, mais uniquement celles qui nous étaient utiles, par exemple lire dans un fichier ne nous intéressait pas.

Les réponses :

Firmware	Disk Selected	Extended Command Set	Short Command Set
All except VDFC	N/A	D:\>↵	>↵ (3E 0D)
VDFC	Source Disk	A:\>↵	>↵ (3E 0D)
	Target Disk	B:\>↵	>↵ (3E 0D)

Tableau 2.1 : tableau des réponses aux commandes

Le *VNC1L*, base du *VDIP2* répond aux commandes via ce qu'on appelle un « *prompt* », en quelque sorte un *ACK*, lorsque tout s'est bien déroulé.

Disk Present	Extended Command Set	Short Command Set
Yes	Voir tableau 2.1	Voir tableau 2.1
No	No Disk↵	ND↵ (4E 44 0D)

Tableau 2.2 : Réponse en cas de non présence de disque

Lorsqu'aucune clé *USB* n'est détectée, le *firmware* renvoi *No Disk<CR>*, ou *ND*<CR>.

Il y a aussi des réponses d'erreurs :

Extended Command Set	Short Command Set	Reason
Bad·Command↵	BC↵ (42 43 0D)	Command not recognised.
Command·Failed↵	CF↵ (43 46 0D)	Filename or directory name not found.
Disk·Full↵	DF↵ (44 46 0D)	No free space on disk.
Invalid↵	FI↵ (46 49 0D)	Attempt to open a directory for reading or writing. Attempt to change currently selected directory to a file.
Read·Only↵	RO↵ (52 4F 0D)	Attempt to open a read only file for writing.
File·Open↵	FO↵ (46 4F 0D)	A file is currently open for writing and must be closed before this command can be executed.
Dir·Not·Empty↵	NE↵ (4E 45 0D)	Attempt to delete a directory which is not empty.
Filename·Invalid↵	FN↵ (46 4E 0D)	Firmware invalid or contains disallowed characters
No·Upgrade↵	NU↵ (4E 55 0D)	Firmware Upgrade file not found on disk

Tableau 2.3 : réponses d'erreurs

La réponse *Bad Command<CR>* est renvoyée lorsque la commande n'est pas reconnue. Ce cas peut arriver, par exemple, lorsque l'on envoie une commande alors que le *VNC1L* est en train d'envoyer une réponse à la commande précédente.

Command Failed<CR>, peut arriver lorsque l'on demande, par exemple, de se positionner dans un répertoire inexistant.

Disk Full<CR> signifie que plus de place n'est disponible sur la clé. Cette erreur arrive lorsque l'on demande de créer un fichier, au moment de fournir sa taille (voir plus bas), mais aussi lorsque l'on interroge le *VNC1L* sur l'espace restant sur le périphérique de stockage.

Read Only<CR> est retourné lors de la tentative d'écriture sur un fichier ayant l'attribut *read only*.

File Open<CR> est retourné lorsque l'on essaie d'ouvrir un fichier, alors qu'un autre n'a pas été fermé.

Dir Not Empty<CR> est retourné lorsque l'on essaie de supprimer un répertoire contenant des fichiers. On doit alors supprimer les fichiers contenus dans celui-ci avant.

Filename Invalid<CR> est retourné lors de la création d'un fichier/dossier, si le nom choisi comporte des caractères invalides : /\ : * ? « » <> |

Les réponses varient de 5 à 17 caractères, en mode étendus, et de deux à trois pour le mode court. D'où le choix pour ce dernier.

Il y a plusieurs types d'instructions :

1) instructions d'initialisation

Extended Command Set	Short Command Set (Hexadecimal Codes)	Function
SCS↵	*10 0D*	Switches to the shortened command set
ECS↵	*11 0D*	Switches to the extended command set
IPA↵	*90 0D*	Monitor commands use ASCII values
IPH↵	*91 0D*	Monitor commands use binary values
SBD·*divisor*↵	*14 20 divisor 0D*	Change monitor baud rate
FWV↵	*13 0D*	Display firmware version
E↵	*45 0D*	Echo 'E' for synchronisation
e↵	*65 0D*	Echo 'e' for synchronisation

Tableau 2.4 : Commandes d'initialisations

Présentées au *tableau 2.4*, les commandes d'initialisations du VNC1L, qui sont les premières commandes à envoyer, après avoir fait un « *scanf* » de 25 ou 60 caractères.

SCS : Permet d'envoyer les commandes SUIVANTES en mode court
ECS : permet d'envoyer les commandes SUIVANTES en mode étendus
IPA : Envoyer les commandes en *ASCII* au *VNC1L*
IPH : Envoyer les commandes en binaire au *VNC1L*
SBD divisor : Changer le baud rate du *VNC1L*, remarque : *divisor* est une valeur a envoyé, voir la table dans le *datasheet*.
FMV : afficher les informations du *firmware* (*scanf* de 60 caractères à prévoir !)

2) Instruction de mise en forme

Extended Command Set	Short Command Set (Hexadecimal Codes)	Function
DIR↵	*01 0D*	List files in current directory
DIR·*file*↵	*01 20 file 0D*	List specified file and size
CD·*file*↵	*02 20 file 0D*	Change current directory
CD·..↵	*02 20 2E 2E 0D*	Move up one directory level
RD·*file*↵	*04 20 file 0D*	Reads a whole file
DLD·*file*↵	*05 20 file 0D*	Delete subdirectory from current directory
MKD·*file*↵	*06 20 file 0D*	Make a new subdirectory in the current directory
MKD·*file*·*datetime*↵	*06 20 file 20 datetime 0D*	Make a new subdirectory in the current directory Also specify a file date and time
DLF·*file*↵	*07 20 file 0D*	Delete a file
WRF·*dword*↵ data	*08 20 dword 0D* *data*	Write the number of bytes specified in the 1st parameter to the currently open file
OPW·*file*↵	*09 20 file 0D*	Open a file for writing or create a new file
OPW·*file*·*datetime*↵	*09 20 file 20 datetime 0D*	Open a file for writing or create a new file Also specify a file date and time
CLF·*file*↵	*0A 20 file 0D*	Close the currently open file
RDF·*dword*↵	*0B 20 dword 0D*	Read the number of bytes specified in the 1st parameter from the currently open file
REN·*file*·*file*↵	*0C 20 file 20 file 0D*	Rename a file or directory
OPR·*file*↵	*0E 20 file 0D*	Open a file for reading
OPR·*file*·*date*↵	*0E 20 file 20 date 0D*	Open a file for reading Also specify a file access date

Extended Command Set	Short Command Set (Hexadecimal Codes)	Function
SEK·*dword*↵	*28 20 dword 0D*	Seek to the byte position specified by the 1st parameter in the currently open file
FS↵	*12 0D*	Returns the free space available on disk if less than 4GB is free
FSE↵	*93 0D*	Returns the free space available on disk
IDD↵	*0F 0D*	Display information about the disk if disk is less than 4GB
IDDE↵	*94 0D*	Display information about the disk
IDD↵	*0F 0D*	Display information about the disk
DSN↵	*2D 0D*	Display disk serial number
DVL↵	*2E 0D*	Display disk volume label
DIRT·*file*↵	*2F 20 file 0D*	List specified file and date and time of create, modify and file access

Tableau 2.5 : Commandes de mise en forme disponibles

Ces commandes permettent de créer un fichier sur la clé *USB*.

CD : permet de se positionner dans un dossier
MKD : permet de créer un fichier
OPW : permet d'ouvrir un fichier afin d'écrire dedans. Si celui-ci n'existe pas, le *VNC1L* le créera pour nous automatiquement.
WRF : écrire dans le fichier ouvert
CLF : fermer le fichier ouvert

Remarques :

1. **Timecode**

Pour *MKD* et *OPW*, un attribut supplémentaire est possible, qui est le *datetime,* qui permet d'horodater un fichier, il s'agit d'un nombre de 32bits comprenant le nombre de secondes écoulées depuis une date de référence, qui est le 1er janvier 1980 pour les systèmes FATS. Ce nombre se calcul de la manière suivante :

Pour 15/07/2010 à 12 :30 :51, avec comme référence : 1er Janvier 1980 minuit.

$Année : 2010 - 1980 = 30 = 0x1E$
$Mois : 7 = 0x7$
$Jour : 17 = 0x11$
$Heure : 12 = 0x0C$
$Minute : 30 = 0x1E$
$Seconde : 51 = 0x33$

Ces valeurs en *Hexadécimal* doivent être placées dans un « ordre » spécial :

Année étant décalé de 25 bits, mois de 21, jours de 16, heure de 11, minute de 5 et seconde n'étant pas décalé.

Un décalage vers la gauche de 25, pour l'année, correspond à multiplier la valeur par 2^{25}, en faisant le même procédé pour toutes les valeurs, on obtient une addition en *hexadécimal*, sous 32 bits, concrètement, en programmation, ça donnera :

```
Datetime = (0x1E << 25) | (0x7 << 21) | (0x11 << 16) | (0x0C << 11) | (0x1E << 5) | (0x30);
```

On aura dans *datetime*, variable 32 bits, une valeur de *0x3CF163F3*.

En le faisant via des variables (year<<25), il faut faire une conversion implicite : (double)(year<<25), puisque 30 décalé de 25 ne tient pas sur le type int déclaré pour year.

2. Arguments de mise en forme

La fonction ***WRF***, requière un argument contenant le nombre de caractères que l'on compte envoyer, pour écrire. Il n'y a pas de caractères de fin de chaîne, ceci permettant d'être transparent, et de transiter tous caractères ASCII.

La fonction étant : « *WRF X<CR>Data* », où X est l'argument précité, et DATA, ce que l'on désire envoyer.

Pour envoyer « *Bonjour* », il faudrait envoyer :
WRF 7<CR>Bonjour

Néanmoins, afin de rendre le projet défensif, nous utilisons la fonction *sprintf*(); dans les *stdio* qui permet de nous renvoyer directement le nombre de caractères composant la chaine que nous désirons envoyer sur le port série. Néanmoins pour fonctionner, nous devons envoyer tous nos caractères et variables dans cette fonction, et cette fonction va placer cette chaîne dans des buffers. Il ne faut pas que le nombre de caractères soit plus grand que le nombre *d'octets* réservés pour ce buffer. Par mesure de sécurité, nous avons doublé l'espace alloué par rapport à ce qu'il devrait y avoir en temps normal.

3. Taille du nom de fichier

Le nombre de caractères est fixé à 8 pour le nom du fichier et de 3 pour l'extension, si un programme envoi 9 ou plus caractères pour le nom, le 9, 10 et 11ème seront comptés comme étant l'extension. Le reste sera tronqué

Exemple de création de fichier

Voici un exemple « *Short Command vs Extended Command* » pour la création d'un même fichier.

Fonction	Mode étendus	Mode court
Prendre les caractères d'initialisation du *VDIP2*	Scanf(''%25c'', &buf) ;	Scanf(''%25c'', &buf) ;
Mode ASCII, et attendre la réponse (en mode étendu = 5 c)	Printf(''IPA\x0D'') ; Scanf(''%5c'', &buf) ;	Printf(''IPA\x0D'') ; Scanf(''%5c'', &buf) ;
Mode approprié (étendu ou court), par défaut ECS	Printf(''ECS\x0D'') ; Scanf(''%5c'', &buf) ;	Printf(''SCS\x0D'') ; Scanf(''%5c'', &buf) ;
Ouverture / création d'un fichier.txt	Printf(''OPW TFE.txt'') ; Scanf(''%5c'', &buf) ;	Printf(''\x09\x20''''TFE.txt\x0D) ; Scanf(''%2c'',&buf) ;
Ecriture dans celui-ci	Printf(''WRF 7\x0D''''Bonjour) ; Scanf(''%5c'', &buf) ;	Printf(''\x08\x20''''7\x0D''''Bonjour'') ; Scanf(''%2c'', &buf) ;
Fermeture du fichier	Printf(''CLF\x0D'') ; Scanf(''%5c'',&buf) ;	Printf(''\x0A\x0D'') ; Scanf(''%2c'', &buf) ;

Le fichier est désormais disponible sur la clé *USB*.

Dans cet exemple, on est en mesure de se demander quel est le choix à faire… Il est vrai qu'avec un exemple aussi simpliste, l'avantage tend vers le mode étendu, plus clair… mais en cas de problèmes, le nombre de caractères renvoyés par le *VNC1L* est différent (jusque 17 !), et le blocage est assuré, bien que dans ce programme, à aucun endroit on vérifie ce que renvoi le *VNC1L*, dans notre programme, c'est bien le cas, on vérifie si une clé est présente, si les fichiers existent… et recevoir toujours 2 caractères est un atout, pour faciliter les tests.

Remarque : on voit au *tableau 2.3* que les réponses en short command sont de 3 caractères, mais on peut en scanner que 2.

3) **Autres**

Le troisième type de commandes étant réservé par exemple, à la commande des ports d'entrées – sorties existants.

Extended Command Set	Short Command Set (Hexadecimal Codes)	Function
IOR·byte↵	29 20 byte 0D	Read I/O Port (1st parameter is port number)
IOW·byte+byte+byte↵	2A 20 byte byte byte 0D	Write I/O Port (1st parameter is port number, 2nd is direction, 3rd is value)

Tableau 2.6 : commandes des ports disponibles

La commande est très facile, soit lire un port (ou un bit, ou plusieurs bits en un coup…), soit écrire un port, un bit ou plusieurs bits en même temps.
Il faut mettre ce tableau en corrélation avec ce tableau suivant :

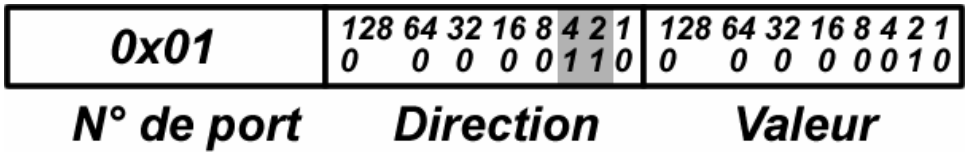
Port Number	Byte 1
AD	0x00
AC	0x01
BD	0x02
BC	0x03 to 0xFF

Tableau 2.7 : Adresse des ports disponibles

Nous avons 4 ports de 8bits, dont certains (bits) ne sont pas disponibles à l'utilisateur (Voir en annexe).

Pour aller allumer notre *LED* bicolore, sur l'ACBUS1 et 2, il suffit de décrire le port ACBUS :

- Bit 1 et 2 en sortie (1= sortie, 0 = entrée)
- Bit 1 = 0, Bit 2 = 1 (allumer Bit1)
- Laisser les autres bits à 0, et en entrée, par exemple



Dans le registre de direction, on précise les bits 1 et 2 en sortie, soit 4+2 = 6, et on allume la *LED* sur le bit 1 = 2

A envoyer : 0x010602

4) Port série

Nous envoyons sur le port série les caractères formant les fichiers que l'on veut créer sur la clé *USB*. Néanmoins, pendant ce temps là, il est impératif d'interdire l'interruption du port série. Les caractères pouvant être reconnus et modifier par exemple un paramètre dans le meilleur des cas, et pouvant faire « planter » le *VDIP2* dans le pire des cas !

5) Timeout

Lorsque l'on effectue un *scanf()*;, le déroulement du programme peut-être compromis, puisqu'il peut-être bloqué par la fonction *scanf()*; en l'attente d'un caractère. Nous avons donc modifié les sources de *getkey()*; et avons ajouté un timeout :

if(timeout)
{
c=2500; //timeout 16MHz 147cy pour 1 bit --> 1174 1 octet --> 2500 par sécurité
while ((!RI)&&c)c--;
}
	Si le flag timeout est à 1, on va attendre un certain temps (tant que C >0), et qu'il n'y a rien sur le port série. S'il n'y a rien, on sort de la fonction.

IV. Eléments rendant notre projet défensif point de vue software et problèmes rencontrés

- Timeout du *Scanf()*;
- Utilisation de la fonction *Sprintf()*; permettant de compter à notre place le nombre de caractères pour la formation du fichier sur la clé *USB*
- Utilisation de l'*EDATA* pour sauvegarder les offsets des *FRAM* mais aussi pour sauver les paramètres
- Détection du premier reset du processeur (pour ne pas écraser les offsets et les données paramétrées)
- Lorsque la tension de la batterie est trop faible, on coupe le chauffage et l'utilisation des moteurs
- Fermeture du couvercle lorsque :
    - L'on tourne le carrousel (à minuit, selon la quantification)
    - L'on ouvre le capteur en position maintenance.
- On connait le nombre d'acquisitions qui ont été faites, même si on introduit une nouvelle heure (PC).
- Vérification si une clé *USB* est présente avant d'écrire dessus
- Bouton prévu pour le chargement des données
- Bouton prévu pour passer en mode DATA (connecter le PC)
- Bouton prévu pour signaler que les filtres ont été changés, afin de ne pas ouvrir le couvercle et contaminer les filtres de la semaine précédente.
- Positionnement du carrousel à l'initialisation
- Positionnement du carrousel lorsque l'on referme le capteur, si besoin (si le faisceau de la barrière infrarouge n'est pas interrompu)
- Sauvegarder tous les logs afin que le technicien puisse se rendre compte de ce qui s'est passé pendant la semaine, et affiner l'algorithme de chauffage
- Pouvoir changer les paramètres via une interface.

Concernant les problèmes rencontrés, ils étaient majoritairement de type *buffer overflow,* le problème *d'echo* avec la fonction *scanf* expliqué dans la partie programmation du *VDIP2*.

Nous avons également eu quelques problèmes avec l'ouverture du couvercle, lorsque l'on rentrait dans une interruption (par exemple *RTC*), toutes les autres étaient bloquées, et de ce fait, la *TIC* ne cadençait plus la lecture du compteur *T0*, lorsque le service des acquisitions synchrones était fini, l'interruption de la *TIC* était effective, et la valeur lue dépendait du temps d'exécution de l'interruption précédente. La valeur du compteur était hors norme, et donc l'ouverture était décidée. Il nous a fallu arrêter de compter à chaque fois que l'on rentrait dans une autre interruption, si l'on était en cadencement des mesures de fréquence.

3. Partie mécanique

Nous n'avons que très peu participé à la partie mécanique, principalement dû à un manque de connaissances. Nous avons été aidés par M. Tarte Pascal.

Nous avons néanmoins aidé à la réalisation de plusieurs parties du projet, celles-ci seront expliquées en détail dans le ce document. Les parties qui ont été réalisées par M. Tarte seront expliquées très brièvement.

La structure du mécanisme

Le mécanisme n'a pas été effectué par nous même, mais nous avons participé quand à la préparation et aux idées principales. Nous sommes partis sur le principe d'un carrousel, comme expliqué juste après, néanmoins, avec Messieurs Tarte et Duvivier, nous avons décidé de placer ce carrousel sur un système coulissant permettant la fermeture facile et hermétique de la porte, lors du changement de filtres, par exemple.

Figure 3.1 : double structure, avec système de coulissage.

Sur la *figure 3.1*, il faut imaginer que la porte sera soudée à la structure interne, faisant pression au niveau de la structure externe. La structure externe a été réalisée d'une récupération d'un TGBT (*Tableau Général Basse Tension*).

Un système de portes à charnière aurait présenté le défaut de ne pas être totalement hermétique aux spores, dans une réalisation artisanale.

Système de changement de filtres

Pour le changement des filtres, M. Duvivier, nous a proposé deux solutions :

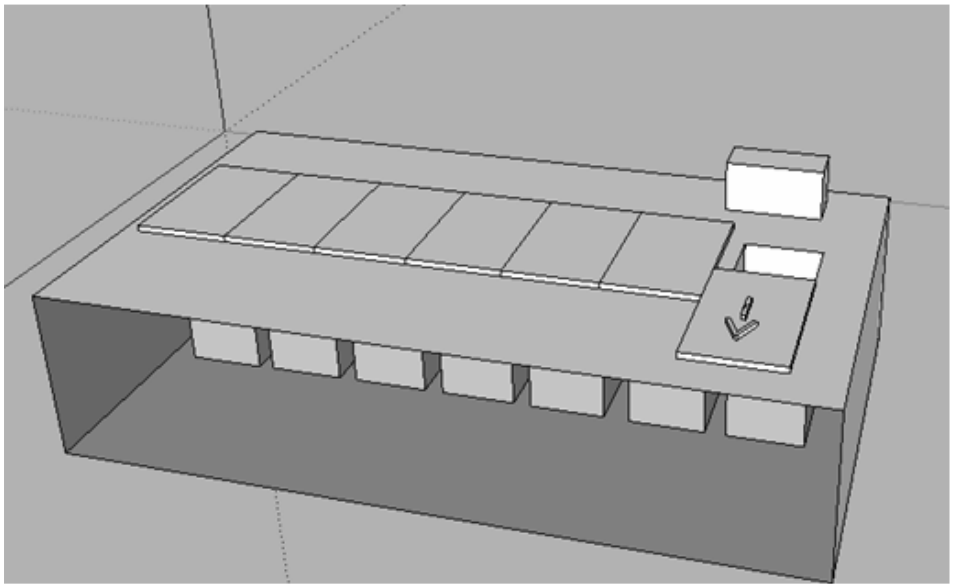


Figure 3.2 : Sept couvercles pour l'échantillonnage

Un système à sept couvercles dont l'un s'ouvre à la fois (dépendant du jour). **L'avantage** réside dans le fait qu'un système de positionnement ne doit pas être mis en œuvre (voir plus bas), et l'on sait nativement où l'on est (pas besoin d'un système de codage absolu gray, par exemple).

Comme inconvénients :
Système mécanique plus complexe (7 couvercles, 7 mécanismes devant être étanches), 7 moteurs, …

Cette solution a été abandonnée.

La deuxième solution, plus viable, est de disposer sept tubes, avec sept filtres, sur un carrousel, actionné par un moteur quelconque, tous les jours à minuit.

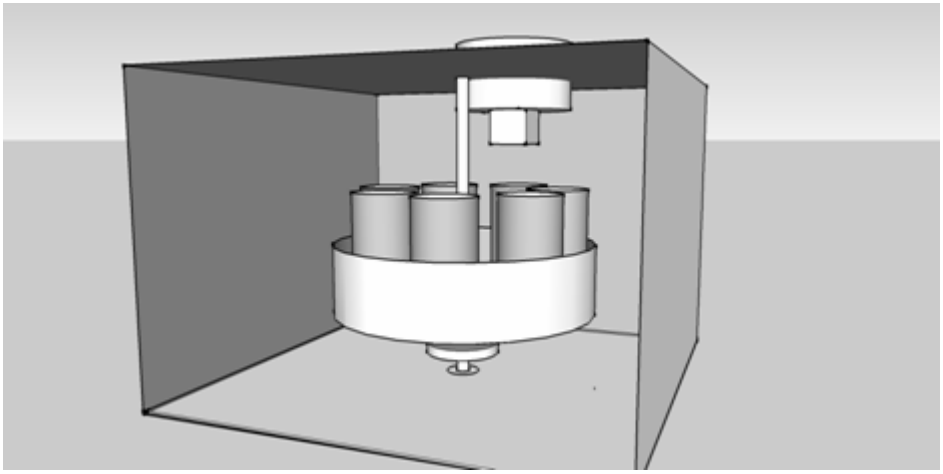


Figure 3.3 : Un seul couvercle pour l'échantillonnage et un carrousel

Nous avons installé 8 tubes, car c'est une puissance de 2, et le huitième sert de témoins, pour quantifier les spores ayant contaminé la boîte, autrement que par la pluie.

Avantage :

- Un seul couvercle
- Besoin d'un seul moteur pour l'unique couvercle
- Espace réduit par rapport au système juste présenté avant
- Electronique plus faible (un seul moteur pour le couvercle)

Inconvénient :

- Besoin d'un système d'axe pour le carrousel
- Besoin d'un système de positionnement (barrière infra rouge, encodeur gray, contact mécanique = usure possible)

Cette solution est retenue, car viable, la réalisation d'un axe étant aisé, et le système de positionnement sera discuté plus bas.

Pour l'axe, nous avons besoin de deux roulements, il existe sur le marché une multitude de types de roulement :

- À billes à contact radial
- À billes à contact oblique
- À rotules sur billes
- À rouleaux cylindriques
- À rouleaux coniques
- À rouleaux sphériques
- À butées à billes
- À aiguilles
- …

Le tout est de savoir quel type de force sera appliqué, et de connaître la conception interne du roulement. Nous allons mettre notre carrousel, et donc notre axe de manière verticale. Il faut donc un roulement acceptant des forces horizontales, dites axiales. De ce fait, on peut déjà exclure les roulements à aiguilles et à galets

Figure 3.4 : Vue de coupe d'un roulement à galet

On peut voir sur la *figure 5*, la vue de coupe d'un roulement à aiguilles, leur forme très allongée leur permettent de supporter des fortes charges radiales, mais n'accepte ***AUCUNE*** charge axiale. Ce qui reviendrait à mettre deux métaux l'un à côté de l'autre et produirait de la friction (pertes, …)

Figure 3.5 : Vue de coupe d'un roulement à billes

Les roulements à billes en général (à une ou à deux rangées de billes), sont conçus pour supporter essentiellement des charges radiales, mais également des charges (plus faibles, certes) axiales, grâce à la profondeur des chemins de roulements. (Forme en croissant de lune de chaque côté, qui épouse la bille). Ces roulements sont les plus utilisés dans l'industrie, pour leur rapport performance/prix. Nous utiliserons ce type de roulement, pour leur grande diversité, et disponibilité dans le commerce.

Les tubes et le système de fixation des filtres

Les tubes ont été dimensionnés pour une pluie inexistante en Belgique, dans l'optique d'éventuellement le vendre à d'autres pays, pour une utilisation en échantillonnage d'eau de pluie. Les filtres de type papier mettent 9.80 secondes (en moyenne) pour évacuer un basculement de pluviomètre (5 ml d'eau).

En regardant les données disponibles au C.R.A. et chez GmE concernant la pluviométrie, on peut s'apercevoir que une forte pluie pour la Belgique est de 40mm sur la journée (en se référant au ***tableau n°1.1*** liant la fréquence d'apparition d'une pluie à sa durée, on peut constater que sur une journée, 43.2 mm d'eau revient tous les 5 ans, en Belgique. Donc, pour 40mm d'eau, on obtient sur la journée 200 basculements pour le modèle de pluviomètre choisi (basculant tous les 0.2 mm)

$$200.5ml = 1000ml = 1l$$

Sur une journée, où un cumul de 40 mm d'eau est enregistré, nous récolterons 1litre d'eau.

Dans ces tubes sera placés un filtre papier, pour filtrant les spores (20µm), ces filtres de 70 mm de diamètre, les tubes feront donc 70 mm de diamètre.

Pour calculer la hauteur du tube, nous devons connaitre la surface :

$$S = \Pi r^2 = \Pi.3,5^2 = 38.49cm^2$$

Sachant que le volume est le produit de la surface par la hauteur, la seule inconnue étant la hauteur. Le volume d'eau est d'un litre, ou 1dm³

$$V = \Pi r^2 h$$

$$h = \frac{V}{\Pi r^2} = \frac{1[dm^3]}{0,385[dm^2]} = 2,59dm = 25,97cm$$

En arrondissant, il faut une hauteur de 26 cm pour contenir une pluie de 40mm, dans nos tubes. Par choix, nous avons pris une hauteur de 45 cm, ce qui permet :

$$V = \Pi r^2 h$$
$$V = 0,385.4,5[dm^2.dm]$$
$$V = 1.73[dm^3]$$
$$V = 1.73[l]$$

On pourrait contenir, sur une journée, une pluie de

$$Volume = \frac{Cumul.5[ml]}{0,2[auget]}$$

$$Cumul = \frac{Volume.0,2}{5} = \frac{1730.0,2}{5} = 69,2mm$$

Si on se réfère au ***tableau 1.1***, une telle pluie revient en moyenne tous les 100 ans, en Belgique, on peut donc dire que notre projet suffira amplement pour l'échantillonnage. Néanmoins, si un cumul supérieur à la contenance est détecté, nous pouvons toujours fermer le couvercle.

Pour fixer les filtres, nous avons établis deux prototypes :

Figure 3.6 : vue des deux systèmes de fixation des filtres

Tous les deux basés sur le même mécanisme à emmancher dans le tube, possédant un pas de vis (voir ***figure 3.7***). Le premier prototype laissait passer l'eau par le coté, vu son trou trop grand (effectué à la foreuse, via une *mèche cloche*). Le deuxième (***figure 3.8.2***) a été découpé soigneusement à la *Dremel*, le résultat est impeccable

Figure 3.7 : Base du système de fixation du filtre

Figures 3.8 : vue des deux systèmes de fixation des filtres

Le premier prototype, comme on peut le voir à la *figure 3.9.1*, déchirait totalement le filtre. Le deuxième le laisse intact (*figure 3.9.2*)

Figure 3.9 : vue des deux systèmes de fixation des filtres

Barrière infrarouge

Nous avons dû mettre la main à la pâte la semaine avant la présentation lors de la journée portes ouvertes de la HELB INRACI. Pendant que M. Tarte réalisait le mécanisme du couvercle, nous avons, mon binôme et moi-même, pensé à cette barrière infrarouge.

Le problème étant que le système initial n'a pas été fait de manière totalement symétrique

Voici ce qui n'a pas été fait, c'est-à-dire découper le cercle en huit parties égales et y apposer les tubes selon un cercle.

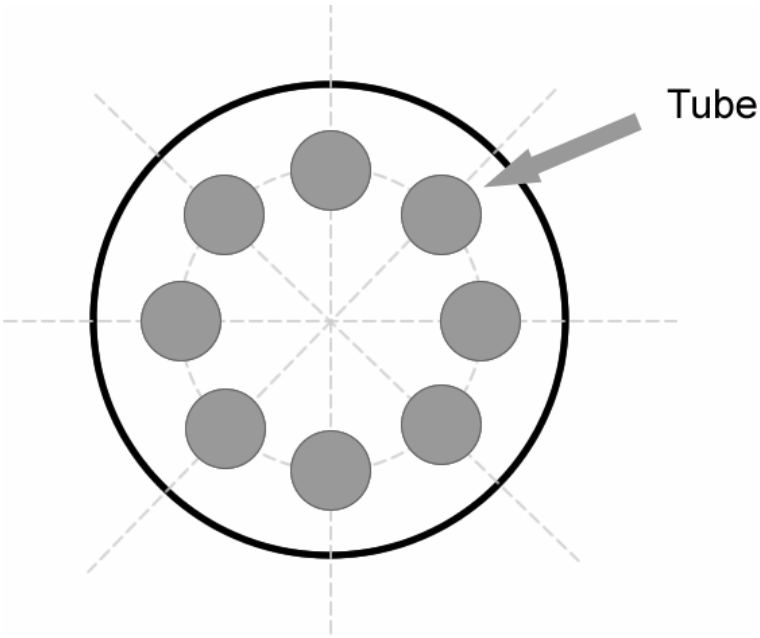


Figure 3.10 : Disposition des tubes sur le carrousel : impossible

Ceci étant tout bonnement impossible, avec le matériel disponible, pour cause, souder 8 pattes à une bague n'est pas du tout aisé.

M. Tarte a émis l'idée de ne souder que 4 pattes (ce qui était possible), et de doubler le nombre de tubes par pattes, comme ceci :

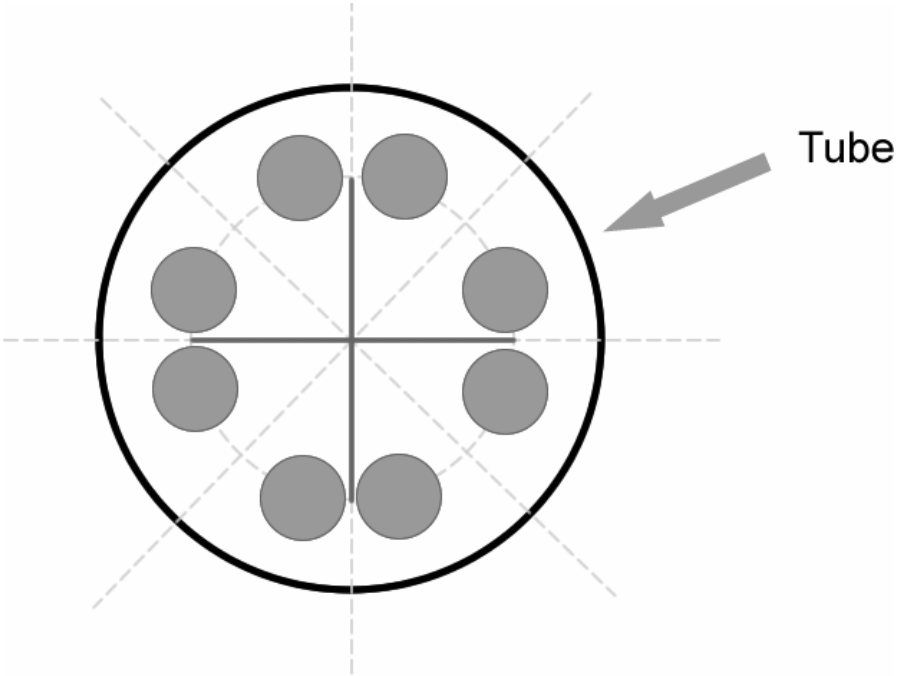


Figure 3.11 : Disposition des tubes sur le carrousel : retenue

Comme on peut le voir, plus aucune symétrie, entre chaque tubes (et donc filtres) consécutifs n'est présente. Lors du design de cette méthode, nous n'avions pas pensé à la barrière infrarouge.

Les tubes étant tenus par des « clips » pour tuyaux PVC, à la réalisation nous avons ceci :

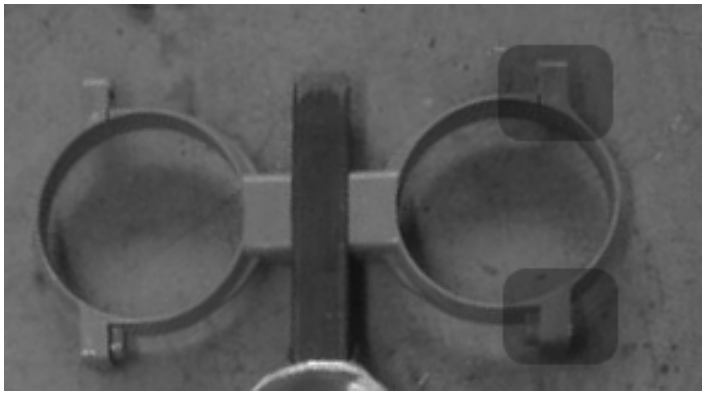


Figure 3.12 : Système de maintient des tubes

Visible à la *figure 3.12*, les deux parties (charnière + verrous) ne sont pas non plus centrées. On ne peut donc pas les utiliser directement pour la détection par la barrière infrarouge… à première vue.

En regardant par après, en adaptant la distance par rapport au centre de ces 4 axes (en bleu), nous arrivons à retrouver une symétrie parfaite.

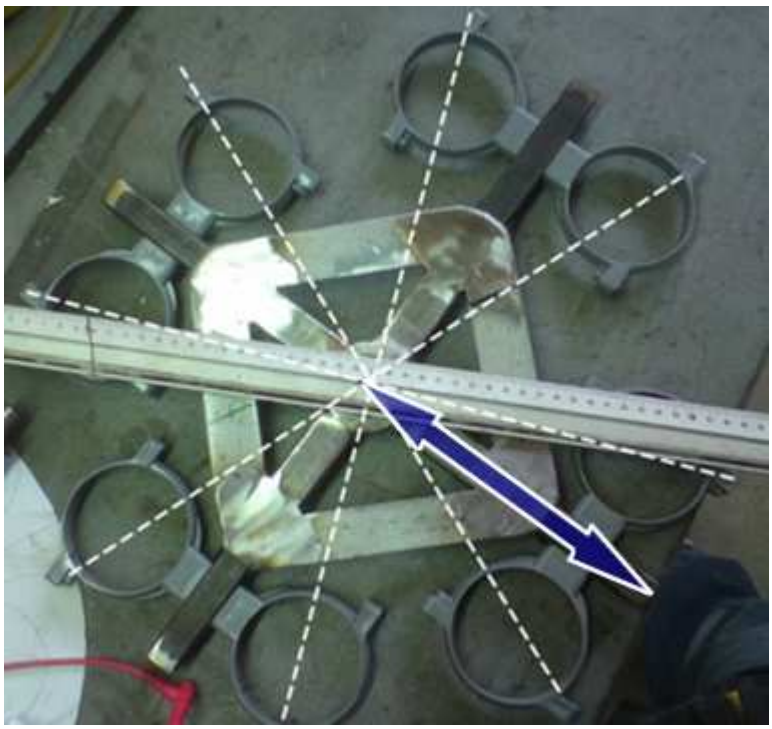


Figure 3.13 : Système de disposition des tubes

Comme le montre la *figure 3.13*, une parfaite symétrie est trouvée. De ce fait, une structure, visible également à la *figure 3.13*, a été rajoutée entre chaque patte, formant un losange. Cette structure permettra d'y fixer 8 vis pouvant interrompre le faisceau infrarouge, au bon endroit (*figure 3.14*).

Figure 3.14 : Système de détection par la barrière infrarouge

Figure 3.15 : Barrière infrarouge

A la *figure 3.15*, on peut apercevoir la barrière infrarouge composée de deux *PCB*, fixée via un système de tige filète, permettant le réglage de la hauteur, et de l'angle de pénétration des visses. Ce système marche parfaitement.

Le panel utilisateur

Nous avons également réalisé le panel. Celui-ci est composé de deux *LED*, deux boutons poussoirs, et deux ports *USB*. Il a été incorporé dans un boîtier choisi après la conception du *PCB*, en effet, au départ nous ne désirions pas en mettre un, pour des problèmes de connectiques (un presse étoupe par câble…). Nous avons néanmoins décidé d'en mettre un, et avons trouvé le boîtier permettant la réalisation directe de celui-ci, visible à la *figure 3.16*, ne nécessitant pas de fils déportés supplémentaires.

Pour les deux ports *USB*, nous avons décidé de récupérer un « *USB Expander* », disponible sur les tours et *desktop* de PC. Ceux-ci s'intégrant parfaitement dans le boitier (*figure 3.17*).

Les boutons choisis sont des SPST (ON)-OFF (*Single Pole, Single Throw*) *Momentary*. Il s'agit de boutons poussoirs NO.

Figure 3.16 : boitier retenus

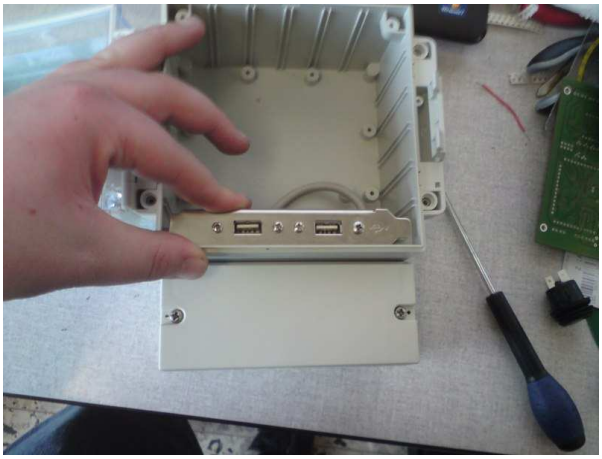


Figure 3.17 : USB Expander

Nous avons dû forer deux trous pour faire tenir le *USB Expander*, et 4 autres pour les boutons et les *LED*

Figure 3.18 : USB Expander placé

Figure 3.19 : Boutons poussoirs placés

Figure 3.20 : Le panel utilisateur Fini

Couvercle

Le mécanisme du couvercle a été fait, au départ, au Génie Rural, mais ce dernier présentant des défauts (frottements), a été refait par M. Tarte.

Lorsque nous avons fait le choix du système de couvercle, nous avons décidé avec M. Duvivier et M. Tarte de faire un système à ouverture horizontale, avec des biellettes, permettant au couvercle de poser par inertie sur le réceptacle du pluviomètre.

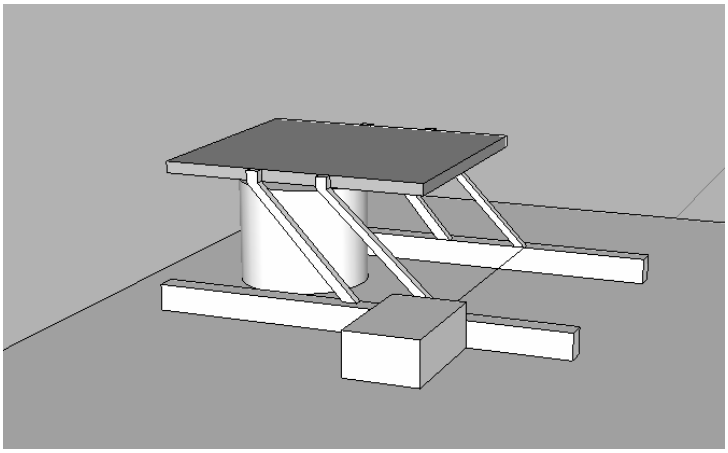


Figure 3.21 : Système de base d'ouverture du couvercle

Les bras de leviers mis en jeux sont très grands, demandant un excédant de couple.

M. Tarte a donc rectifié le mécanisme, en y incorporant le moteur à l'intérieur de la boîte visible à la *figure 3.22*, et en y insérant également le capteur d'humectation, avec une pente de 30°.

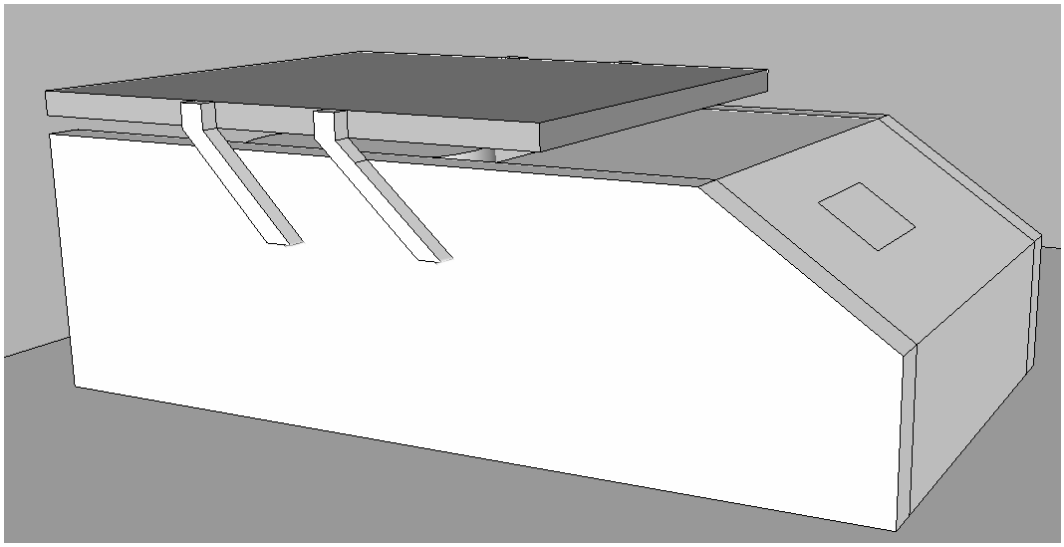


Figure 3.22 : Système d'ouverture du couvercle retravaillé

Les avantages sont que les bras de leviers sont moindres, et que l'électronique interne est protégée. Le moteur se trouve à l'intérieur et entraine un des axes des bras de leviers via un système de pignons et de chaîne, à démultiplication nulle.

Problèmes rencontrés

Servomoteur

Le Servomoteur du couvercle a été choisi identiquement à celui du carrousel. Pour le carrousel, le couple était suffisant (et même excessif, voir partie programmation). Par contre, pour le couvercle, le couple disponible était bien trop faible. Le *Datasheet* de *Futaba* nous renseignant 0.3 Nm possible. Nous avons emprunté au Génie Rural un dynamomètre, permettant de connaître le couple nécessaire (Force * bras de levier)

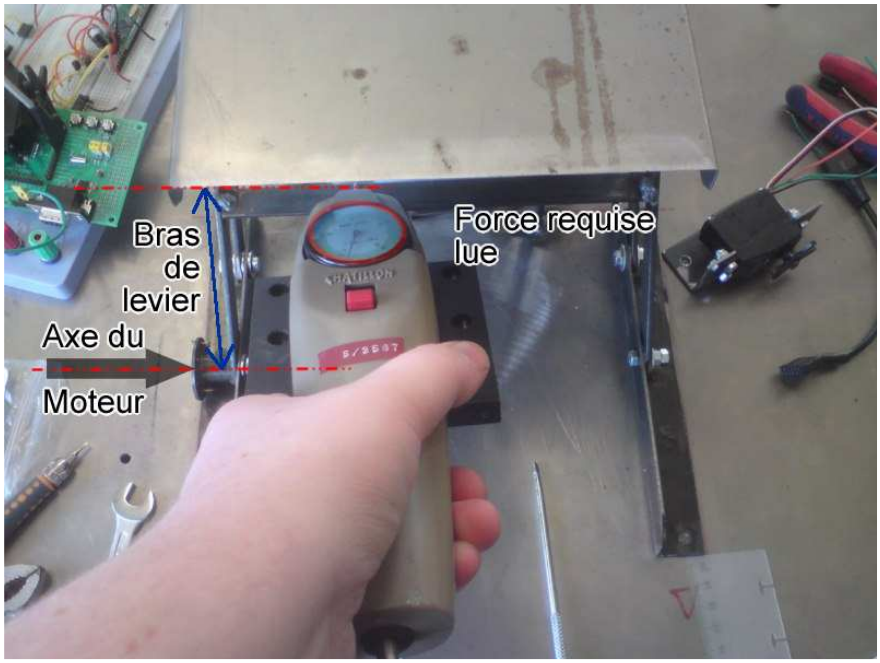


Figure 3.23 : Exemple illustrant la mesure de couple

A la *figure 3.23*, une mesure du couple requis est démontrée. Le Dynamomètre à ressort mesurant la force nécessaire à la mise en mouvement du couvercle (dans notre cas).
En relevant le bras de levier, par rapport à l'axe du moteur, on peut déterminer le couple requis. Dans notre cas, il fallait en moyenne 1.54 Nm (voir *tableau 3.1*), et avons opté pour le *FUTABA* S9157 proposant 3Nm de couple à 6V.

Force (kg)	Bras de levier (cm)	Couple (Nm)
1,5	9	1,32435
0,89	18,5	1,6152165
2,1	6	1,23606
1,6	9	1,41264
1,5	13,5	1,986525
0,8	25	1,962
1,7	7,5	1,250775
	MOYENNE :	**1,54108093**

Tableau 3.1 : relevé de couple

4. Tests réalisés

Nous n'avons pas encore pu réaliser un test en grandeur nature, en cause la partie mécanique non finie, néanmoins, plusieurs tests ont été effectués.

Dans un premier temps, nous avons testé chaque module du projet séparément (surtout le *VDIP*1 et le *VDIP2*), en validant au jour le jour leur fonctionnement.

Progressivement, nous avons assemblé ces modules ensemble (tant bien programmation que électronique, sur *project board*), et également validé les étapes.

Ensuite, nous avons fait des acquisitions durant toute une nuit avec les différents capteurs, dont les résultats étaient directement enregistrés sur la clé *USB*.

Lorsque le *PCB* nous a été livré, et quand il a été soudé, nous avons mis chaque nuit sous tension le projet, en y laissant l'écriture des logs, ce qui nous a permis, chaque jour de détecter des bugs et de les corriger (problèmes de *buffer overflow*, à plusieurs endroits)

Nous avons également testé notre capteur d'humectation, d'abord en laboratoire (faire bouillir de l'eau dans une étuve), et voir l'effet de la condensation sur celui-ci. Nous avons pu déterminer de ce fait un bon algorithme de chauffage (expliqué dans la partie programmation), qui tient compte à la fois de la température par rapport au point de rosée, mais aussi de la fréquence.

Lorsque cet algorithme donnait de bons résultats en laboratoire, nous l'avons testé et laissé dehors plusieurs nuits, et nous avons pu valider cet algorithme directement.

Nous avons également testé le *sensirion*, dans un boitier étanche, allant de 0% d'HR à 100% d'HR (ou 75% avec du *NAcl*), la variation est linéaire, et progressive, il possède une grande hystérèse (plusieurs heures). Le *sensirion* est étalonné en usine.

5. Mode d'emploi

Mise en service :

Installer le capteur de spores sur une batterie 12V.

Eléments à vérifier :

- le coulissage de la structure interne, au besoin graissez
- le bon fonctionnement du mécanisme du couvercle, au besoin graissez
- Vérifiez la tension de la batterie
- Vérifiez la connexion entre le processeur et le PC, en cas d'échec, vérifiez :
    - Que la tension de la batterie est bien au niveau du bornier
    - Que la procédure de connexion a été respectée (juste ci dessous)
- Lorsque la connexion aura abouti, mettez à jour le système (heure/date), vérifiez vos paramètres, …
- Vérifier que le capteur de détection de pluie soit propre, au besoin dégraissez
- Placez huit nouveaux filtres dans les huit tubes
- Placez un nouveau sachet de *silica gel*.
- Si toutes ces étapes sont correctes, fermer le capteur, à clé, revenez dans une semaine.

Connexion entre le PC et la carte

Veuillez à toujours respecter ces étapes :

1) Ouvrez le capteur, avec la clé, et tirer la porte
2) Ouvrez le boîtier, et insérez le câble *USB* dans le port adéquat (celui sur votre gauche, lorsque vous êtes en face), n'insérez rien dans le deuxième port *USB* !
3) Appuyez sur le bouton de droite, la *led* en dessous initialement verte va passer rouge, pour redevenir vert, le microprocesseur est prêt.
4) Insérez l'autre bout du câble *USB* dans votre ordinateur
5) Lancez « *CRA.exe* »
6) Appuyez sur « *Check for Ports* » (surligné en rouge à la *figure 4.1*), cette application va vérifier les ports de communication disponibles, si une erreur se produit, veuillez retourner à l'étape 1.
7) Lorsque le système a trouvé tous vos ports de communication, veuillez appuyer sur « *Find µP* » (surligné en bleu à la *figure 4.1*), cette procédure permettra de distinguer sur quel *COM* est connecté le projet, lorsque votre ordinateur possède plusieurs ports *COM*.

Figure 5.1 : Interface

8) Si le système ne trouve pas l' « *ADµC* », veuillez recommencer depuis l'étape 1. Lorsque le système a trouvé le microprocesseur, vous pouvez paramétrer le capteur à votre convenance.

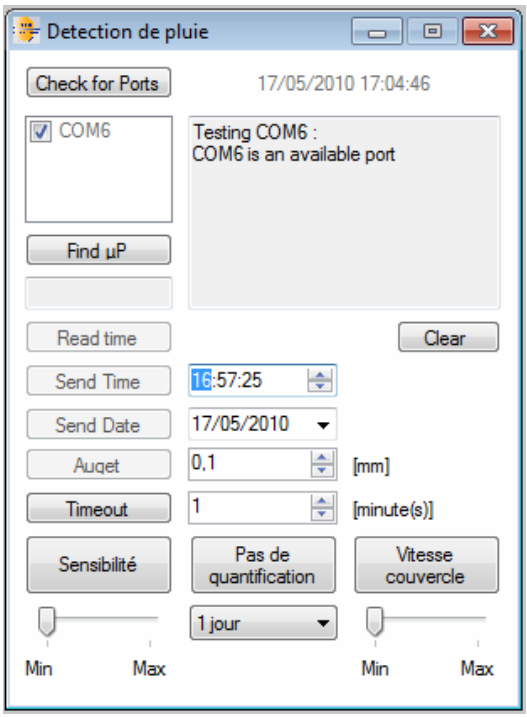


Figure 5.2 : L'interface utilisateur prête

Parfois, il se peut que le programme *CRA.exe* ne fonctionne pas (notamment si vous utilisez Linux, Mac Os, …), le paramétrage reste néanmoins totalement possible, mais fait appel à un personnel qualifié.

Rem : veillez toujours à respecter les 4 premières instructions pour établir une connexion.

1) Ouvrir le capteur, avec la clé, et tirer la porte
2) Ouvrir le boîtier, et insérer le câble *USB* dans le port adéquat (celui sur votre gauche, lorsque vous êtes en face), n'insérez rien dans le deuxième port *USB* !
3) Appuyez sur le bouton de droite, la *led* en dessous initialement verte et va passer rouge, puis vert, le microprocesseur est prêt.
4) Insérez l'autre bout du câble *USB* dans votre ordinateur
5) Ouvrez un terminal (Sur Windows: HyperTerminal, à éventuellement télécharger, vérifiez au préalable), entrez un nom de communication (sans importances)
6) Choisissez le bon port *COM,* définissez 9600 baud, 8 bits de données, pas de parité, un bit d'arrêt et pas de contrôle de flux.
7) Désormais, vous avez accès aux mêmes fonctionnalités, mais sans l'interface graphique. Vous devez utiliser les mêmes commandes, telles que définies dans le tableau suivant :

Fonction	A envoyer	Réponse	Lié ?
Vérifier la disponibilité du processeur	F	OK	Oui, à toutes
Lire l'heure et la date du capteur	R	OK	Oui, à « A »
-	A	Aduc time : dd/mm/aaaa – hh:mm:ss	Non
Envoyer la Date au capteur	D + Date*	-	Non
Envoyer l'heure au capteur	T + heure**	-	Non
Faire tourner le carrousel	G	« end » quand positionné	Non
Ouvrir le couvercle	O	« end » quand ouvert	Non
Fermer le couvercle	f	« end » quand fermé	Non
Paramétrer la sensibilité	S+1 ou 0***	Ok avant l'envoi de la sensibilité	Non
Vitesse d'ouverture / fermeture du couvercle	V + 1 ou 2 ou 3 ****	Ok avant l'envoi	Non
Timeout du pluviomètre, en minutes, de **0**1 à 99 min	t+TEMPS	Ok	Non
Modifier le type d'auget	A+auget*****	Ok	Non
Nombre de jours par filtres (1 à 7)	Q+Nbrs de jours	Ok	non

Tableau 5.1 : commandes de paramétrage

Lié signifie que cette fonction entame le processus d'une autre, par exemple, pour lire la date/heure du projet, il faut faire « F », puis « R », et enfin « A).

* = Il faut envoyer F, puis D suivit directement (sans espaces) de jj/mm/aaaa
** = pareil, sauf que il faut envoyer l'heure dans le format : hh:mm:ss
*** = 0 = peu sensible, 1 = fortement sensible. Seuls ces deux modes existent, tout autre chiffre donnera fortement sensible
**** = 1 = rapide, 2= peu rapide, 3 = lent. Tout autre chiffre donnera lent
***** = Envoyer le type de pluviomètre (exemple 0.2), multiplié par 10, soit 02 dans l'exemple. Attention, il est important de mettre le 0 devant si un seul chiffre.
Rem : le F n'est à faire qu'une fois, mais peut-être fait au début de chaque fonction

6. Bref rappel des protocoles utilisés

Le Bus I2C

Bus développé par Phillips pour des applications domotique et électronique début 1990. Celui-ci permet de relier facilement des périphériques à un microprocesseur.

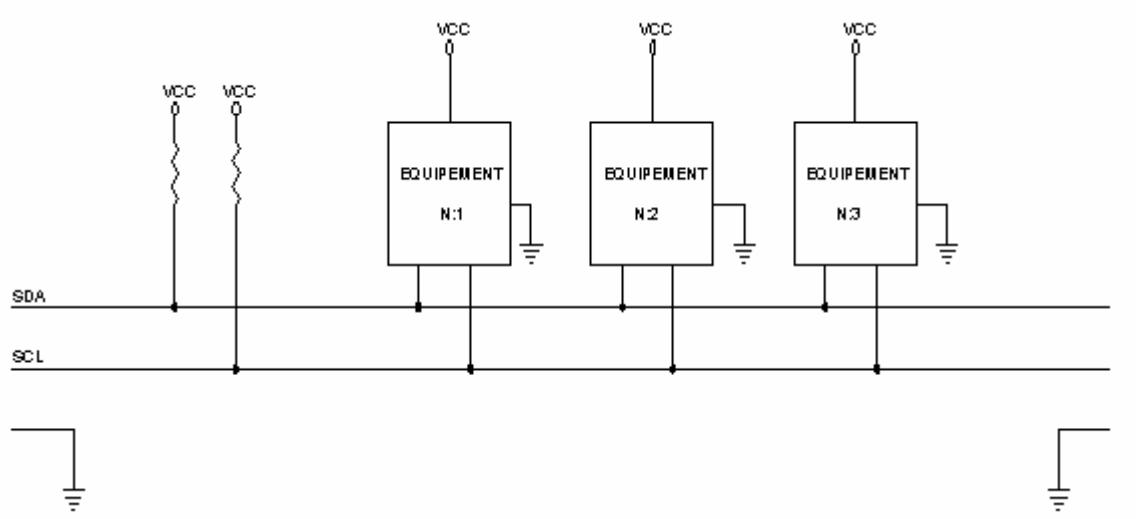


Figure 6.1 : Exemple d'interfaçage d'une installation en I2C

Ce bus est composé de deux fils : *SCLOCK et SDATA* (en plus de la masse).

Il s'agit d'un bus synchrone (la vitesse de transmission peut varier, puisque cadencé par SCL), les limites d'utilisation du bus I2C dépend uniquement de la capacité parasite (intégration) entre SDA / SCL et la masse, celle-ci ne devant pas dépasser 400pF. Néanmoins on peut diminuer la fréquence de transmission pour augmenter le nombre d'équipements sur le bus. Chaque périphérique ajoutant sa capacité parasite.

L'adresse d'un périphérique est différente s'il l'on demande d'écrire un octet dans celui-ci, ou si l'on demande de lire, en effet, le dernier bit de l'octet réservé à l'adresse correspond à :

- *0 = écriture*
- *1 = lecture*

L'adresse est codée sur 7 bits, donc théoriquement jusque 128 circuits disponibles sur un bus (hormis les adresses réservées).

On peut interfacer sur un bus I2C des Master (µC/µP), des interfaces générales (telles que afficheurs LCD, ports d'entrées sorties, …) mais aussi des circuits orientés application spécifiques (Radio, GSM,…)

Ce bus gère le *multi master*, mais nous avons utilisé le bus I2C uniquement en mode *Master Slave*, donc seul ce principe sera rappelé.

Le bus I2C possède sa propre série de signaux de contrôle, tels que :

- *Stop Bit*
- *Start Bit*
- *Repeted start Bit*
- *(Not) Acknowledgment*
- …

Le master (notre *ADµC*), prend l'initiative sur le BUS I2C, et via un système d'adressage demande à un périphérique particulier de lui fournier une donnée.

Exemple de communication :

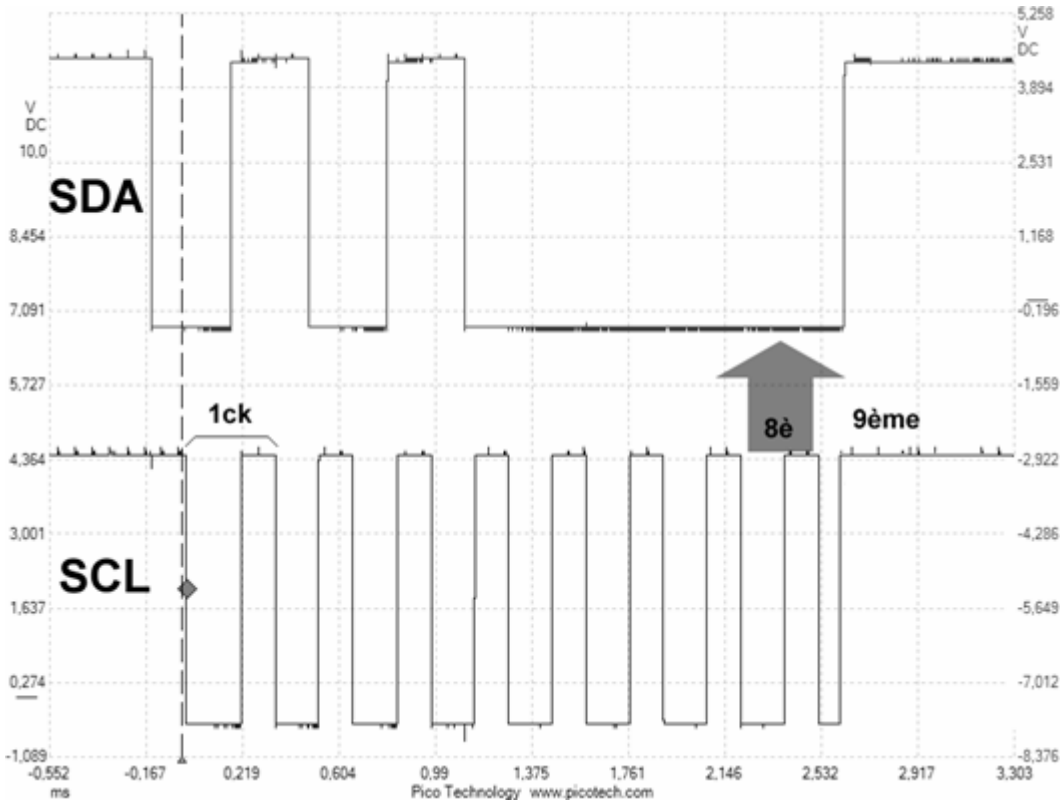


Figure 6.2: Communication entre l'ADµC et la RTC.

Ici, nous avons une procédure d'écriture sur notre *RTC* (Adresse 0xA0). En fait, nous avons écris l'adresse de l'offset à partir duquel on désire lire les données.

Nous devons écrire pour faire une lecture !

Le bit de start = SCL flanc descendant, puis SDA flanc descendant.

8 Bits cadencés par SCL, et 8 bits de données.

On voit bien que le 8ème bit du SDA est bas (=écriture). Le 9eme bit (qui est en fait 1.5bits) est un bit STOP (état haut sur SCL puis SDA, jusqu'au prochain bit start).

Le bus USB

Introduction

Figure 6.3 : logo de la norme USB

Conçu au milieu des années 1990, afin de remplacer les nombreux ports externes d'ordinateurs lents et incompatibles. Plusieurs versions de la norme furent développées au fur et à mesure des avancées technologiques

1. ***USB* 1.0 :** permettant des vitesses de 1.5 Mbit/s (*Low speed*), et 12 Mbit/s (*Full Speed*).
2. ***USB* 1.1** : correction de la 1.0 avec vitesses identiques
3. ***USB* 2.0** : Vitesse de communication jusque 480 Mbit/s (*High speed*)
4. ***USB* 3.0** : Permettant une vitesse de 4,8 Gbit/s (*Super Speed)*

Rem : L'*USB* permet la compatibilité ascendante, puisque les connectiques sont toutes identiques.

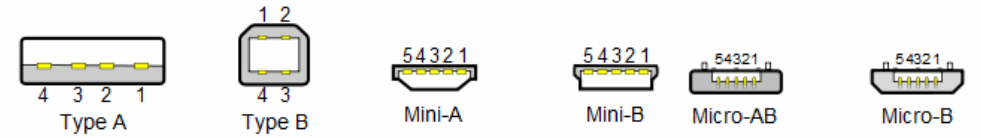


Figure 6.4 : les différents connecteurs de la norme

Fonction	Couleur	Numéro de broche pour les types A et B	Numéro de broche pour le type mini B
Alimentation +5V (VBUS)	Rouge	1	1
Données (D-)	Blanc	2	2
Données (D+)	Vert	3	3
Masse (*GND*)	Noir	4	5

Tableau 6.1 : Brochage des connecteurs, se rapportant à la figure 4.4

L'*Universal Serial Bus* (*USB*) est donc un bus de transmission série synchrone (le plus souvent), parfois asynchrone, à haute et très haute vitesse, qui permet d'interconnecter plusieurs périphériques à un seul hôte. En effet, la différence avec l'I2C, par exemple, abordé juste avant, est que le *Multi Master* n'est pas géré.

L'*USB* autorise le *Hot Swap*, qui est une connexion et déconnection dit « à chaud », donc en cours de fonctionnement. Il n'est pas nécessaire de couper les périphériques avant de les débrancher. Généralement les périphériques *USB* sont autoalimentés via le port, grâce aux lignes d'alimentation *VBUS* et la masse, le courant maximal admissible est de 500mA.

Pour les données, il y a deux fils, *D+ et D-*, formant une paire torsadée, garantissant une immunité aux bruits.

Figure 6.5 : Paire torsadée USB

Le protocole

Ce protocole est basé sur le principe de l'interrogation successive de chaque périphérique. Lorsque l'hôte désire communiquer avec un périphérique, il envoie un paquet de données (susnommés jeton), comportant l'adresse du périphérique destinataire, codé sur 7 bits. Lorsqu'un périphérique reconnait son adresse dans ce jeton, il répondra par un « *ACK* », et la communication peut s'établir. Bien que codé sur 7 bits, seuls 127 périphériques sont disponibles, car l'adresse 0 est une adresse « tampon », qu'obtient un périphérique connecté à chaud, avant que l'hôte lui attribue une adresse.

La communication utilise un jeu de commandes permettant à l'hôte de spécifier le contenu des données envoyées :

- **Un paquet « Jeton »**
  - *JETON IN :* Informe le périphérique que l'hôte veut lire des données
  - *JETON OUT :* Informe le périphérique que l'hôte veut envoyer des données
  - *JETON SETUP :* Utilisé pour commencer un transfert de commande

- **Un paquet de données :** Contient jusque 1024 octets, et commence toujours par le LSB

- **Un paquet d'état** ou *HandShake*
  - **ACK**
  - **NACK**
  - **STALL (***bloqué***) :** Lorsque l'appareil se retrouve dans un état qui va exiger l'intervention de l'hôte.

Types de transferts

4 types sont prévus dans cette norme :

1. **Les transferts de commandes**
   Utilisé pour les opérations de commandes et d'états (énumération d'un périphérique : fournir une adresse et détecter le type de périphérique). Ce mode de transmission est garanti : renvoie des paquets perdus.

2. **Les transferts d'interruptions**
   Utilisé pour fournir des petites informations avec un temps de latence très faible (comme pour un clavier, une souris, …). Le périphérique doit néanmoins attendre que l'hôte l'interroge. Ce n'est donc pas une interruption au sens informatique du terme.

3. **Les transferts isochrones**
   Utilisé pour des transferts volumineux en temps réel (bande passante est garantie). Il n'y a pas de garantie de transfert. Utilisé dans l'acheminement de flux audio et vidéos.

4. **Les transferts en masse (*bulk*)**
   Utilisé pour les transferts volumineux, avec garantie d'acheminement, mais sans garantie de bande passante. Utilisé pour les stockages de masse.

Le codage utilisé dans cette norme est le *No Return to Zero Inverted* (*NRZI*) :

- Un « 0 » induit une transition
- Un « 1 » n'en induit pas.

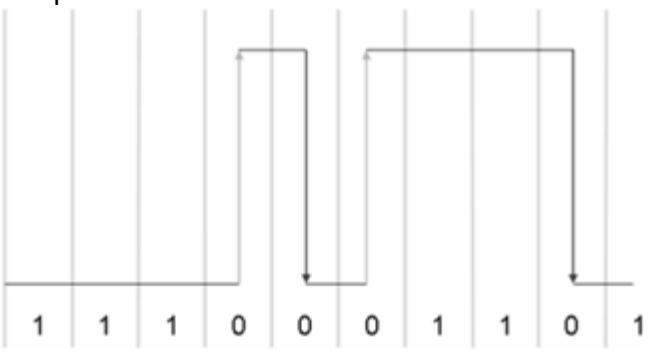


Figure 6.6: exemple de communication NRZI

Il faut noter que dans notre projet, ce bus est géré de manière interne et transparente par notre *VNC1L* de *FTDI*.

L'UART

Universal Asynchronous Receiver Transmitter (*UART*), est un composant clé permettant la liaison entre périphériques. Un système asynchrone n'est donc pas cadencé par une horloge commune entre récepteur et émetteur, il faut donc pouvoir admettre une petite tolérance.

Un UART est couplé à un microprocesseur, par exemple 8bits pour le bus de données. Nous allons devoir créer une conversion parallèle série via des registres à décalages (LSB envoyé en premier).

Au repos, le système laisse sa sortie série haute. Lorsque celui-ci veut envoyer à un périphérique une donnée, il émet un bit de *start* (niveau bas). Ensuite vient les 8 bits de données, et enfin le bit de *stop* (haut). Ce bit de stop peut parfois faire 1, 1.5 ou 2 temps de bits.

La taille des données est généralement comprise entre 5 et 9 bits.

Il est également prévu d'envoyer un bit de parité, selon la logique parité paire ou impaire.

Figure 6.7 : Trame UART

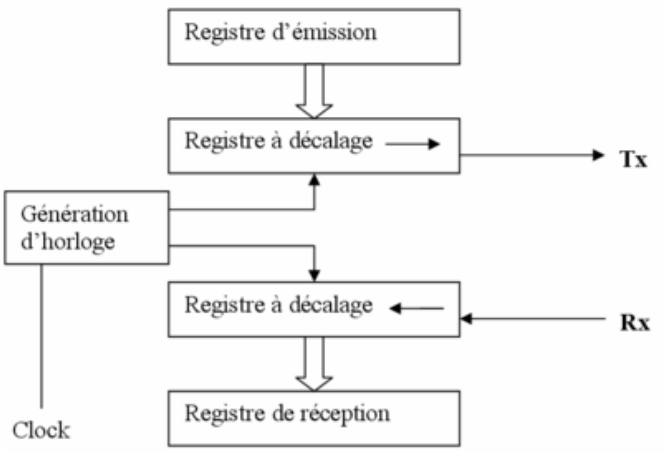


Figure 6.8 : Schéma bloc simplifié de l'UART

Le microcontrôleur va commencer l'envoi en introduisant dans le registre d'émission. La conversion parallèle série peut s'opérer selon une cadence définie (voir plus bas). Lorsque l'envoi est terminé, un *flag* préviendra le microcontrôleur que le registre d'émission est vide, et qu'il peut désormais renvoyer un autre octet. Ceci est fait pour éviter de monopoliser du temps CPU, en effet l'envoi d'un octet prend à peu près 1 ms (10bits à 9600 bits/s).

Pour la réception, c'est pareil, on reçoit en série, *l'UART* reconstitue un octet via son registre à décalage, lorsque la transmission est cohérente, *l'UART* place cet octet dans le registre de réception, et en informe le microcontrôleur via un autre *flag*. Ce dernier dispose aussi de 1ms (par exemple) pour aller lire ce registre, sans quoi la donnée est perdue. Ce registre joue le rôle de tampon.

Il y a plusieurs vitesses normalisées :

- 110 bps
- 300 bps
- 1 200 bps
- 2 400 bps
- 4 800 bps
- **9 600 bps**
- 19 200 bps
- 38 400 bps
- 57 600 bps
- 115 200 bps
- 230 400 bps

9600 bits/s (ou bps pour *bit per second*) est la vitesse de transmission par défaut.

Rem : Généralement la vitesse de transmission série est exprimée en baud/s, qui défini un débit de moment (nombre de symboles transmis par seconde). **Dans le cas de *l'UART***, un baud transporte un bit, et donc baud/s = bit/s

7. Améliorations possibles et prévues

Prévues :

Nous avons prévu un deuxième IO Expander PCF8574 permettant la disponibilité de 8 ports d'entrées sorties supplémentaires.

Nous avons également prévu deux emplacements pour le capteur de pluie IBR273.

Possibles :

- Redessiner le *PCB*, afin de corriger les erreurs trouvées
- Changement du système de mise en mouvement du carrousel :
    - ✓ Moteur pas à pas

      En effet, un moteur pas à pas permet de bloquer une position avec du couple, tout en offrant une précision et la possibilité de tourner plus d'un tous.

    - ✓ Garder un Servomoteur, mais changer le rapport de démultiplication

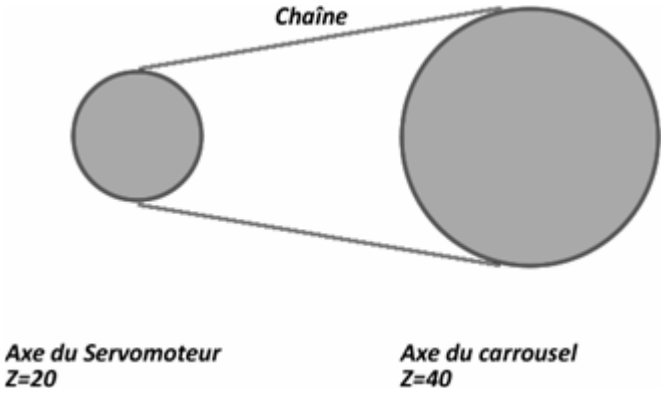


Figure 7.1 : Système de démultiplication présent entre le Servomoteur et l'axe du carrousel

En effet, pour le moment nous avons cette configuration, avec un rapport de 2 entre les nombres de dents sur le pignon de sortie su Servomoteur, et sur celui présent sur l'axe du carrousel. Lorsque le moteur effectue un tour, un demi-tour est réalisé au niveau du carrousel. Ceci avait été choisi pour diminuer le couple nécessaire au Servomoteur. Si l'on prend ce principe à l'inverse, et qu'on agrandit le pignon sur l'axe du Servomoteur, dans un rapport de deux, nous aurons avec une dynamique de source de 180°, une dynamique finale de 360°. Ce qui serait parfait, même au point de vue du couple, puisque l'axe étant sur roulements, quasiment peu de couple est requis.

- Utilisation d'un compteur externe pour la mesure de fréquence de l'astable. En effet une telle mesure aurait permis de mettre l'*ADµC* en full *power down* (sauf lors d'acquisitions et autres évènements « spéciaux »), réduisant la consommation de

manière significative. Le fabriquant annonce 20µA de consommation pour l'*ADµC* 832, dans ce mode de fonctionnement.

- Pour détecter la pluie, et l'évacuer correctement, notre capteur IBR273 doit être placé à un peu près 30° d'inclinaison :

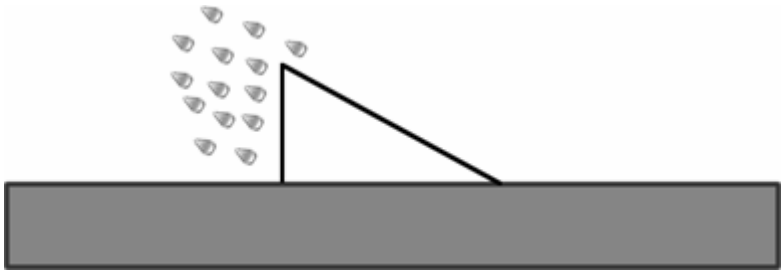


Figure 7.2 : Vue simplifiée de l'implantation du capteur de détection de pluie

Or, il se peut que le vent dévie l'orientation des goutes, comme montré à la ***figure 53***, et que les gouttes raflent le capteur. Bien sûr il y en aurait quand même une qui tomberait dessus, mais suivant la sensibilité, le temps pourrait être plus long.

L'idée serait de mettre le capteur sur un roulement, avec une girouette qui le positionnerait toujours de manière optimale face au vent.

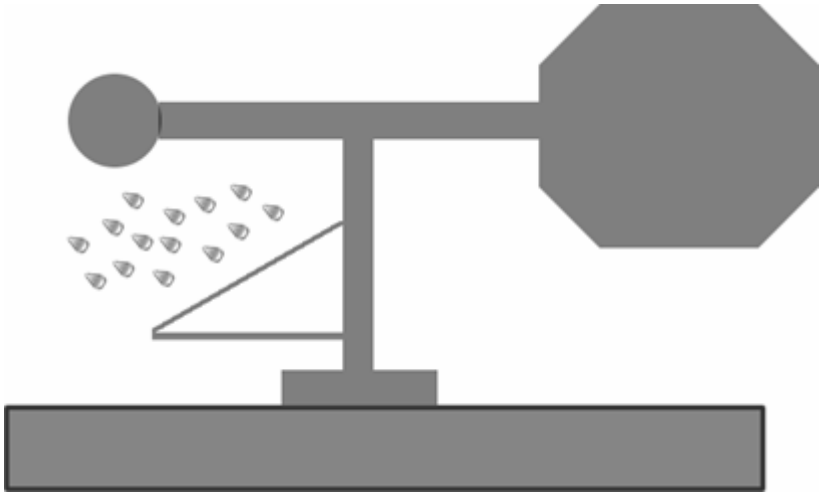


Figure 7.3 : Système de positionnement du capteur de détection de pluie par rapport au vent

La contrainte liée à ce système serait qu'il faudrait six contacts tournants (système de six cercles avec six picots)

- Switch permettant de connaître la position du couvercle à tout moment, évitant de l'ouvrir s'il est déjà ouvert

8. Liste des composants et devis

Comme susmentionné, tous les composants ont été commandés chez Farnell (sauf mention contraire).

#	réf farnell	Qty	Description	PAU	PAT
1	1678545	1	*LED* IR	€ 0,55	€ 0,55
2	1243889	1	Photo Transistor	€ 0,25	€ 0,25
3	1299914	2	CTN 10k@25℃ Betatherm	€ 0,64	€ 1,28
4	1077336	1	socket DIL40	€ 2,90	€ 2,90
5	1234656	1	TLC556 : DUAL TIMER 555	€ 1,19	€ 1,19
6	8576378	2	monting clip RING *LED* 5mm	€ 0,22	€ 0,44
7		2	Microswitch fin de course NO	€ 0,23	€ 0,46
8	1634672	2	Bouton poussoir SPST IP56	€ 2,25	€ 4,50
9	1702623	1	battery holder 9V	€ 1,62	€ 1,62
10	1142565	1	*LED* Yellow low consumption	€ 0,10	€ 0,10
11	1019939	1	*USB* 2,0 type A	€ 3,42	€ 3,42
12	1644426	1	INDUCTOR 0,51A SMD	€ 2,10	€ 2,10
13	1414664	10	CAPA 0805 100nF	€ 0,04	€ 0,39
14	9227806	2	CAPA 0805 1µF 16V	€ 0,08	€ 0,16
15	1463377	1	capa 0805 10µF 6,3V	€ 0,21	€ 0,21
16	9693793	2	Capa TH 100µF 50V	€ 0,13	€ 0,26
17	1288279	4	capa 0805 470nF	€ 0,06	€ 0,24
18	1142568	1	*LED* GREEN/RED 5mm	€ 0,44	€ 0,44
19	1692688	1	inductor 220µH	€ 3,56	€ 3,56
20	1625035	2	Diodes shottky ES2B	€ 0,57	€ 1,14
21	1617737	2	diodes switching 1n4248	€ 0,06	€ 0,12
22	8648689	2	MOSFET P -55V/-17A	€ 1,03	€ 2,06
23	1621736	1	BRIDGE RECTIFIER 3A/50V	€ 0,78	€ 0,78
24	1399737	4	10E Resistor 0805	€ 0,03	€ 0,13
25	1500691	2	220E resistor 0805	€ 0,48	€ 0,96
26	1576182	1	9,1M Resistor 0805	€ 0,05	€ 0,05
27	1577666	1	3,9K Resistor 0805	€ 0,69	€ 0,69
28	1690393	2	I/O EXPANDER PCF8574	€ 1,78	€ 3,56
29	1013958	1	74HC14 inverter	€ 0,43	€ 0,43
30	403908	1	PCF8583 : Real Time Clock	€ 4,08	€ 4,08
31	1750145	2	Ampli OP LM324	€ 0,23	€ 0,46
32	1688869	2	RAMTRON NV-FRAM 64kbits IIC	€ 15,33	€ 30,66
33	1621017	2	74HC74 D FlipFlop	€ 0,32	€ 0,64
34	1460664	1	Switching regulator 3A ADJ LM2576	€ 3,38	€ 3,38
35	1273683	1	Switching regulator 5V Fixed LT1111-5	€ 5,93	€ 5,93
36	1675765	3	Pin header - lockable - 2,54mm 3WAY M	€ 0,26	€ 0,78
37	1675755	3	Pin header - lockable - 2,54mm 3WAY F	€ 0,12	€ 0,36
38	EUROCIRCUITS	1	*PCB* Spores C010 - V2,07	€ 28,30	€ 28,30
40	1023037	1	SOCKET IC, WIREWRAP, 8WAY	€ 0,89	€ 0,89
41	1590513	1	SENSOR, HUMIDITY & TEMP, V4	€ 28,63	€ 28,63
42	1653445	2	TACTILE SWITCH	€ 0,30	€ 0,60
43	9333720	10	RESISTOR, 0805 10K	€ 0,03	€ 0,32
44	9333711	10	RESISTOR, 0805 1K	€ 0,03	€ 0,32
45	9509682	2	CRYSTAL, 32.768000KHZ	€ 0,94	€ 1,88

46	LEXTRONIC	1	Senseurs IBR273	€	4,56	€	4,56
48	1079400	1	ADUC 832	€	17,70	€	17,70
49	9556192	7	BS170	€	0,25	€	1,75
50	rs: 429-280	1	*VDIP2*	€	27,23	€	27,23
51	Conrad : 384348-62	1	*Futaba* S3003	€	10.45	€	10.45
52	Conrad : 207423-62	1	*Futaba* S9157	€	120.69	€	120.69
			TOTAL	€	295.54	€	323.60
			Autres:				
	100012	1	Pluvio PR12	€	122,45		
-		1	boitier	€	35,00		

Tous ces prix s'entendent hors TVA.

Le prix de revient, pour la partie électronique est de 446.05 € HTVA, soit 539.72€ TVAC.

Pour comparaison, le prix de revient (192.46€ HTVA pour une carte) passe à 144.83€ HTVA par carte, pour 50 unités produites.

Pour la partie mécanique, M. Tarte estime à 120 heures la réalisation par unités. Néanmoins, ce nombre d'heure est à nuancer en fonction de l'équipement disponible. S'il avait eu beaucoup plus de moyens technique, M. Tarte estime à 80 heures le temps nécessaire à la réalisation complète.

Concernant la conception mécanique, le chiffrage est plus délicat, puisque nous étions à 4 pour décider et réfléchir. A ce titre, nous estimons à 65 heures le temps nécessaire qu'il nous a fallu pour la conception.

Et enfin, nous ne compterons pas moins de 30 heures en temps d'essais mécanique, c'est-à-dire, le temps que M. Tarte a pris pour tester différentes configurations, et d'usinage de pièces non utilisées car le résultat ne nous convenait pas.

Concernant les matières premières, la plupart des tôles provient de récupération (*TGBT*). Les profiles étaient également sur place. Néanmoins, nous avons du faire une partie de la mécanique au génie rural, ceux-ci ont facturés 500€ pour le système d'axe, 5 tôles (soudées), et le mécanisme du couvercle (qui a dû être refait par M. Tarte).

9. Conclusion

Pour conclure, je vais commencer par l'énumération succincte de la résolution des grands points du cahier des charges.

a. La détection de pluie rapide se fait via la mesure d'une variation de capacité, lors de l'apport d'eau sur un capteur.

b. L'ouverture du couvercle hermétique se fait via un servomoteur

c. Le changement de filtres/tubes s'opère à minuit (ou au choix de l'utilisateur). Le positionnement se fait via une barrière infrarouge et un encodeur de position absolue à codage GRAY

d. Afin d'être le meilleur marché possible, nous avons pris en compte à chaque fois le ratio prix/consommation et avons réalisé deux *PCB* nous-mêmes.

e. Afin d'être robuste, le capteur en général a été réalisé en acier traité à l'antirouille et nous avons adopté des *RAM* non volatiles en cas de coupure de courant (batterie HS). Notre électronique est placée dans un boîtier avec du *silicagel* à l'intérieur.

f. La consommation (mesurée) de la carte, hors chauffage, moteurs et chargement *USB* est toujours inférieure à 10mA, nos régulateurs étant à découpages.

g. Les données sont récupérables via un support de type « clé *USB* ».

Le travail demandé a donc été réalisé, et ce dans les délais (14 semaines). Nous n'avons pas encore réalisé de tests grandeur nature, car la mécanique, au jour de l'impression de ce TFE n'est pas encore finie. M. Duvivier ne sait pas encore dire si la pertinence de la captation des spores est suffisante, et si oui dans quelle mesure. Cette étude étant encore vierge de documentations, ce TFE est plus à caractère expérimental que la mise en place d'un raisonnement existant.

Durant ce stage, j'ai appris à réagir rapidement et efficacement lorsqu'un problème est décelé. De pareilles situations ont permis d'affiner mon esprit d'analyse.

En trois années de formation à la HELB INRACI, je crois vraiment avoir gagné en maturité par rapport à ma sortie de mes A2 Techniques (en électronique). Lors de ma sortie de rhétorique, je ne me sentais pas capable de réaliser un tel travail.

In fine, je suis assez fier du travail que nous avons accompli, et espère grandement que les résultats seront à la hauteur des attentes de M. Duvivier et M. Moreau, je les remercie vivement pour ce sujet passionnant. Je suis également content d'avoir gagné la confiance d'une institution, puisque le CRA souhaite nous engager, mon binôme et moi-même en tant que *jobiste* durant le mois d'août, pour fabriquer un deuxième capteur de spores.

10. Bibliographie et annexes

Ma bibliographie est exclusivement numérique

- Caleca (Christian), La FAT16, exemple de gestion de fichiers, 02/04/2006, Marseille, http://www.stielec.ac-aix-marseille.fr/cours/caleca/pc/fat16.html

- *Le Pévédic* (*Jérémie*), Guidage en rotation, les types de roulement, 18/10/2000, s.l, http://guidageenrotation.free.fr/roultype.php3

- X, Techno-Sciences.net : La permittivité, s.d, s.l, http://www.techno-science.net/?onglet=glossaire&definition=3292
- X, Wikipédia, FAT 16 (les), 2007-2009, s.l, http://fr.wikipedia.org/wiki/FAT16

- X, Wikipédia, File Allocation Table, 2007-2010, s.l, http://fr.wikipedia.org/wiki/File_Allocation_Table

- X, Wikipédia, l'I²C, 2007-2010, s.l, http://fr.wikipedia.org/wiki/I'I²C

- X, Wikipédia, NE555 (le), 2009-2010, s.l, http://fr.wikipedia.org/wiki/NE555

- X, Wikipédia, permittivité (la), 2007-2010, s.l, http://fr.wikipedia.org/wiki/Permittivité

- X, Wikipédia, Le pluviomètre, 2005-2010, s.l, http://fr.wikipedia.org/wiki/Pluviomètre

- X, Wikipédia, Relation de Steinhart-Hart, 2005-2009, s.l, http://fr.wikipedia.org/wiki/Relation_de_Steinhart-Hart

- X, Wikipedia, Universal Serial Bus, 2004-2010, s.l, http://fr.wikipedia.org/wiki/Universal_Serial_Bus

- Les différents *datasheet* provenant des différents fabriquant, de nos différents circuits

- Cours dispensés durant ces trois années à la HELB - INRACI

Les sources du programme, tous les *datasheet*, le schéma de principe, les organigrammes, et les *PCB* sont présent sur le CD-ROM ci-joint dans ce TFE.